Популярные лекции
по математике

В. А. УСПЕНСКИЙ

МАШИНА ПОСТА

ПОПУЛЯРНЫЕ ЛЕКЦИИ ПО МАТЕМАТИКЕ
ВЫПУСК 54

В. А. УСПЕНСКИЙ

МАШИНА ПОСТА

ИЗДАНИЕ ВТОРОЕ,
ПЕРЕРАБОТАННОЕ

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1988

ББК 22.12
У77
УДК 510.6(023)

Успенский В. А.

У77 Машина Поста. — 2-е изд., испр. — М.: Наука. Гл. ред. физ.-мат. лит., 1988. — 96 с. — (Популярные лекции по математике).

ISBN 5-02-013735-9

Машина Поста — это хотя и абстрактная (т. е. не существующая в арсенале действующей техники), но зато очень простая вычислительная машина. Она способна выполнять лишь самые элементарные действия, и потому ее описание и составление простейших программ может быть доступно ученикам начальной школы. Тем не менее на машине Поста можно запрограммировать — в известном смысле — любые алгоритмы. Изучение машины Поста можно рассматривать как начальный этап обучения теории алгоритмов и программированию.

1-е изд. — 1979 г.

Для широкого круга читателей.

У $\frac{1702020000-063}{053(02)-88}$ 60-88

ББК 22.12

Владимир Андреевич Успенский

МАШИНА ПОСТА

Серия «Популярные лекции по математике», выпуск 54

Редактор *В. В. Донченко*
Художественный редактор *Т. Н. Кольченко*
Технический редактор *Л. В. Лихачева*
Корректоры *М. Л. Медведская, Н. Б. Румянцева*

ИБ № 32624

Сдано в набор 22.05.87. Подписано к печати 12.01.88. Формат 84×108/32. Бумага книжно-журнальная. Гарнитура литературная. Печать высокая. Усл. печ. л. 5,04. Усл. кр.-отт. 5,25. Уч.-изд. л. 5,37. Тираж 89 000 экз. Заказ № 595. Цена 20 коп.

Ордена Трудового Красного Знамени издательство «Наука»
Главная редакция физико-математической литературы
117071 Москва В-71, Ленинский проспект, 15

Ленинградская типография № 2 головное предприятие ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 198052. г. Ленинград, Л-52, Измайловский проспект, 29.

ISBN 5-02-013735-9

© Издательство «Наука», Главная редакция физико-математической литературы, 1979; с изменениями, 1988

ОГЛАВЛЕНИЕ

Предисловие 4

Глава первая

КАК РАБОТАЕТ МАШИНА ПОСТА

§ 1. «Внешний вид» машины Поста 7
§ 2. Программа машины Поста 10
§ 3. Работа машины Поста 12
§ 4. Примеры выполнения программ 14
§ 5. Методические замечания 17

Глава вторая

ПРИБАВЛЕНИЕ ЕДИНИЦЫ НА МАШИНЕ ПОСТА

§ 1. Запись чисел в машине Поста и постановка задачи о прибавлении единицы 22
§ 2. Прибавление единицы в простейшем случае 24
§ 3. Прибавление единицы в более сложных случаях 27
§ 4. Прибавление единицы в еще более сложном случае . . 31
§ 5. Прибавление единицы в самом общем случае 36

Глава третья

АНАЛИЗ И СИНТЕЗ ПРОГРАММ МАШИНЫ ПОСТА

§ 1. Диаграммы и блок-схемы 37
§ 2. Анализ программы прибавления единицы 41
§ 3. Еще о задаче прибавления единицы 46
§ 4. Сложение чисел в простых случаях 50
§ 5. Сложение чисел в более сложных случаях 55

Глава четвертая

ВОЗМОЖНОСТИ МАШИНЫ ПОСТА

§ 1. О задаче сложения чисел на произвольных расстояниях 58
§ 2. Предложение (постулат) Поста 61
§ 3. Машина Поста и алгоритмы 66
§ 4. Вычислимые функции и тезис Поста 71
§ 5. Машина Поста и электронные вычислительные машины 77

Приложение 1. Финитные комбинаторные процессы, формулировка 1. *Эмиль Л. Пост* 83
Приложение 2. Решимые и нерешимые алгоритмические проблемы. *В. А. Успенский, А. Л. Семенов* 89

ПРЕДИСЛОВИЕ

Эта книга рассчитана прежде всего на школьников. У читателя первых трех глав не предполагается почти никаких математических знаний, выходящих за рамки начальной школы. Думается, что содержание первых двух глав доступно даже ученикам младших классов.

В книге излагается некоторая «игрушечная» (по-научному — «абстрактная») вычислительная машина — так называемая машина Поста, вычисления на которой отражают многие существенные черты вычислений на реальных компьютерах; на элементарных примерах происходит обучение началам программирования для машины Поста и выясняются возможности этой машины, которые, несмотря на ее предельную простоту, оказываются довольно большими.

Автор надеется, что предлагаемая книга в какой-то мере сможет способствовать продвижению в среднюю школу — и даже в ее начальные классы — таких понятий, как «алгоритм», «универсальная вычислительная машина», «программирование». Личный опыт автора убеждает его, что школьники первых классов и даже старшие дошкольники без труда могут осуществлять «вычисления»[1]) на машине Поста по заданной программе, например, при помощи разграфленной на секции бумажной ленты и канцелярских скрепок или пуговиц в качестве меток, а также составлять простейшие программы (не содержащие команд передачи управления). Именно поэтому в первую, наиболее доступную для младших школьников главу включен специальный, пятый параграф «Методические замечания».

---

[1]) Слово «вычисления» взято в кавычки, потому что вовсе не обязательно, чтобы исходные данные и результаты осуществляемых на машине преобразований были числами; в ряде случаев гораздо нагляднее операции с сочетаниями символов, не имеющих числового значения.

* * *

Свыше пятидесяти лет назад выдающийся американский математик Эмиль Л. Пост опубликовал в «Журнале символической логики» статью «Финитные комбинаторные процессы, формулировка 1» (ее перевод составляет приложение к основному тексту настоящей книги). В этой статье и в появившейся одновременно в «Трудах Лондонского математического общества» статье английского математика А. М. Тьюринга «О вычислимых числах с приложением к проблеме разрешения» были даны первые уточнения понятия «алгоритм» — одного из центральных понятий математической логики и информатики, играющего все более и более важную роль в вопросах автоматизации, а поэтому и во всей жизни современного общества.

Уточнения понятия «алгоритм», предложенные Постом и Тьюрингом, не теряют своего значения до нашего времени. Машина Тьюринга постоянно используется в качестве рабочего аппарата в современной теории алгоритмов; машина Поста менее популярна, хотя — или именно по той причине, что — она проще машины Тьюринга [1]).

Указанные работы замечательны еще и тем, что в них за несколько лет до появления первых действующих экземпляров больших (так называемых «универсальных») вычислительных машин (сперва даже не электронных, а электромеханических) были в абстрактной форме предвосхищены основные принципиальные черты таких машин. Сами конструкции были предложены Постом и Тьюрингом в виде неких «абстрактных машин»; это сделано в явной форме Тьюрингом и в неявной — Постом, у которого термин «машина» отсутствует. Изложение построений Тьюринга часто приводится в литературе по теории алгоритмов, в том числе и популярной. Что же касается построений Поста, то, несмотря на их даже бóльшую, чем тьюринговские, простоту, они, за исключением оригинальной статьи Поста, долгое время не излагались ни в специальной, ни в популярной

[1]) Машина Поста устроена проще, чем машина Тьюринга, в том отношении, что ее элементарные действия проще, чем элементарные действия машины Тьюринга, и способы записи менее разнообразны, однако именно по этим причинам запись и переработка информации на машине Поста требует большего объема «памяти» и большего числа шагов, чем на машине Тьюринга.

литературе[1]). В то же время именно эти построения, как показывает опыт осуществлявшегося автором преподавания, могут составить не менее, чем машины Тьюринга, естественное введение в теорию алгоритмов.

Предложенному Постом понятию алгоритма и посвящена настоящая книга. Изложение несколько модифицировано и, в частности, проводится в форме описания некой абстрактной вычислительной машины. Именно поэтому кажется целесообразным введение самого термина «машина Поста» и вообще использование отличающейся от постовской терминологии.

Основу этого, 54-го, выпуска «Популярных лекций по математике» составляют лекции автора для школьников (в Московском университете в октябре 1962 г., в школе-интернате при Новосибирском университете в июне 1963 г., в Московском Дворце пионеров в декабре 1964 г.), а также лекции для студентов механико-математического и филологического факультетов МГУ — начиная с 1961/62 учебного года.

По сравнению с первым изданием, вышедшим в 1979 г., в брошюру внесены следующие изменения: во-первых, полностью обновлен § 4 главы IV, во-вторых, добавлено Приложение 2, в-третьих, внесены немногочисленные локальные исправления. Оба Приложения можно читать независимо друг от друга и от остального текста книги.

---

[1]) В 1967 г. в журнале «Математика в школе», в № 1—4, автором была опубликована серия из четырех статей, положенных в основу данного издания.

ГЛАВА ПЕРВАЯ

КАК РАБОТАЕТ МАШИНА ПОСТА

§ 1. «Внешний вид» машины Поста

Прежде всего предупредим читателя, что машина Поста не есть реально существующее, сделанное кем-то устройство; поэтому слова «внешний вид» и взяты в кавычки. Машина Поста, как и ее близкий родственник — машина Тьюринга,— представляет собой мысленную конструкцию, существующую лишь в нашем воображении [1]) (хотя ее в принципе и можно было бы изготовить «в металле» [2])). Именно это имеют в виду, когда говорят о машинах Поста и Тьюринга, что они суть «абстрактные» вычислительные машины. Однако для нас будет несущественным, что машины Поста на самом деле нет. Напротив, мы будем предполагать ее «как бы существующей» — для наглядности. И подобно тому, как можно выучиться считать на счетах или на логарифмической линейке, не имея перед собой этих приборов, а пользуясь лишь их описаниями и представляя их себе мысленно, так же и мы научимся вычислять на машине Поста, прилагая наше воображение к тому ее описанию, которое сейчас будет дано.

Машина Поста состоит из *ленты* и *каретки* (называемой также *считывающей и записывающей головкой*).

---

[1]) Поэтому анекдотически выглядят такие, например, рекомендации относительно ознакомления с машинами этого типа: «для того, чтобы лучше представить действительную работу машины, интересующийся должен обратиться к технической литературе» (Попов А. И. Введение в математическую логику. — Л.: Изд-во ЛГУ, 1959. — С. 91).

[2]) Устройство, позволяющее моделировать работу машины Поста в случае небольших программ и небольших объемов вычислений, изготовлено в 1970 г. в Симферопольском государственном университете — см.: Касаткин В. Н. Семь задач по кибернетике. — Киев: Вища школа, 1975. — С. 26.

Лента бесконечна и разделена на *секции* одинакового размера: для наглядности ленту будем считать расположенной горизонтально (рис. 1).

Бесконечность ленты находится в противоречии со сделанным выше утверждением, что машину Поста можно было бы в принципе построить. Дело в том, что мы объявили ленту бесконечной лишь для простоты изложения. С тем же успехом можно было бы предположить, что лента не бесконечная, а лишь *неограниченно*

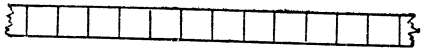


Рис 1. Лента машины Поста разделена на секции и неограниченно простирается влево и вправо

растущая в обе стороны: например, можно было бы считать, что лента наращивается на одну секцию, как только каретка доходит до конца ленты и должна двигаться дальше (о движении каретки смотри ниже), или считать, что за каждую единицу времени слева и справа нарастает по одной секции. Нам, однако, будет удобнее считать, что все секции слева и справа уже наросли, и тем самым, хотя и в ущерб реальности, полагать ленту бесконечной в обе стороны.

Порядок, в котором расположены секции ленты, подобен порядку, в котором расположены все целые числа. Поэтому естественно ввести на ленте «целочисленную

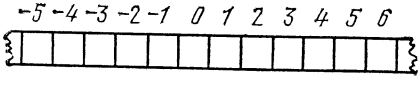


Рис. 2

систему координат», занумеровав секции целыми числами ..., —3, —2, —1, 0, 1, 2, 3, ... (рис. 2).

Мы будем считать, что система координат жестко сопоставлена с лентой, и получим таким образом возможность указывать какую-либо секцию ленты, называя ее порядковый номер, или координату. (Иногда, впрочем, бывает удобно наряду с основной, «постоянной» системой координат, ввести еще вспомогательную, «временную», сдвинутую по отношению к первоначальной.)

В каждой секции ленты может быть либо ничего не записано (такая секция называется *пустой*), либо

записана *метка* V (тогда секция называется *отмеченной*) (рис. 3).

Информация о том, какие секции пусты, а какие отмечены, образует *состояние ленты*. Иными словами, состояние ленты — это распределение меток по ее секциям [1]). Как мы далее увидим, состояние ленты меняется в процессе работы машины.

Каретка может передвигаться вдоль ленты влево и вправо. Когда она неподвижна, она стоит против ровно

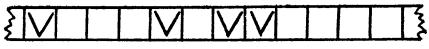


Рис. 3. В каждой секции ленты либо не записано ничего, либо записана метка

одной секции ленты (рис. 4, *а*; на этом и следующих чертежах каретка изображена в виде зачерненного квадрата); говорят, что каретка *обозревает* эту секцию, или держит ее в *поле зрения*.

Информация о том, какие секции пусты, а какие отмечены и где стоит каретка, образует *состояние машины*

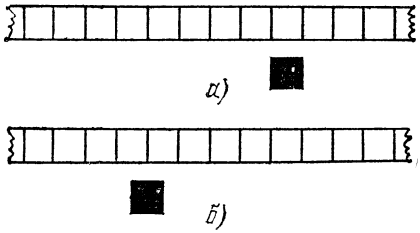


Рис. 4. Когда каретка неподвижна, она стоит против одной из секций ленты так, как показано на рис. *а*), а не так, как показано на рис. *б*). Ситуация, изображенная на рис. *б*), может возникнуть только в процессе движения каретки

Поста. Таким образом, состояние машины слагается из состояния ленты и указания номера той секции, которую обозревает каретка. За единицу времени (которую мы будем называть *шагом*) каретка может сдвинуться на одну секцию влево или вправо. Кроме того, каретка может поставить (напечатать) или уничтожить (стереть) метку в той секции, против которой она стоит,

[1]) На точном математическом языке состояние ленты — это функция, которая каждому числу (номеру секции) ставит в соответствие либо метку, либо, скажем, слово «пусто».

а также распознать, стоит или нет метка в обозреваемой ею секции. Чем определяются действия каретки, а также, что значит «распознать» в применении к каретке, будет объяснено в § 3.

§ 2. Программа машины Поста

Работа машины Поста состоит в том, что каретка передвигается вдоль ленты и печатает или стирает метки. Эта работа происходит по инструкции определенного вида, называемой программой. Для машины Поста возможно составление различных программ. Посмотрим, как устроена программа.

Каждая программа машины Поста состоит из команд. *Командой машины Поста* будем называть выражение, имеющее один из следующих шести видов (буквы i, j, j_1, j_2 означают всюду натуральные числа 1, 2, 3, 4, 5, ...):

Первый вид. Команды движения вправо.

$$i. \Rightarrow j$$

Второй вид. Команды движения влево.

$$i. \Leftarrow j$$

Третий вид. Команды печатания метки.

$$i. \vee j$$

Четвертый вид. Команды стирания метки.

$$i. \xi j$$

Пятый вид. Команды передачи управления.

$$i. \ ? \begin{cases} j_1 \\ j_2 \end{cases}$$

Шестой вид. Команды остановки.

$$i. \text{ стоп.}$$

Например,

$$137. \Rightarrow 1$$

является командой движения вправо,

$$25. \ ? \begin{cases} 32 \\ 25 \end{cases}$$

— командой передачи управления, а

6386. стоп

— командой остановки.

Число i, стоящее в начале команды, называется *номером* команды. Так, у приведенных только что команд номера суть соответственно 137, 25 и 6386. Число j, стоящее в конце команды (а у команд передачи управления — каждое из чисел j_1 и j_2), будем называть *отсылкой* (при этом в команде передачи управления j_1 — верхней, а j_2 — нижней отсылкой). У команд остановки нет отсылки. Так, у приведенных только что команд отсылками служат числа 1, 32, 25, причем 32 — верхняя отсылка, а 25 — нижняя отсылка.

Программой машины Поста будем называть конечный непустой (т. е. содержащий хотя бы одну команду) список команд машины Поста, обладающий следующими двумя свойствами:

1) На первом месте в этом списке стоит команда с номером 1, на втором месте (если оно есть) — команда с номером 2 и т. д.; вообще на k-м месте стоит команда с номером k.

2) Отсылка любой из команд списка совпадает с номером некоторой (другой или той же самой) команды списка (более точно: для каждой отсылки каждой команды списка найдется в списке такая команда, номер которой равен рассматриваемой отсылке).

Например, следующий список будет программой машины Поста:

1. стоп
2. ? 4 / 1
3. §3
4. стоп.

А эти два списка не будут программами машины Поста, хотя и составлены из команд машины Поста:

2. ? 4 / 1
3. §3
1. стоп
4. стоп

(не выполнено первое условие);

1. стоп 3. §3

2. ? $\begin{matrix} \diagup 4 \\ \diagdown 5 \end{matrix}$ 4. стоп

(не выполнено второе условие).

Для наглядности программы машины Поста мы будем записывать столбиком. Число команд программы называется *длиной* программы.

Упражнение. Выпишите все программы машины Поста длины 1. Сколько существует программ длины 2, длины 3, длины n?

§ 3. Работа машины Поста

Чтобы машина Поста начала работать, надо задать, во-первых, некоторую программу, а во-вторых, некоторое ее (машины) состояние, т. е. как-то расставить метки по секциям ленты (в частности, можно все секции оставить пустыми) и поставить каретку против одной из секций. Как правило, мы будем предполагать, что в начальном (т. е. в задаваемом вначале) состоянии машины каретка ставится всегда против секции с номером (координатой) нуль. При таком соглашении начальное состояние машины полностью определено состоянием ленты.

Как уже говорилось, программа является той инструкцией, на основании которой работает машина. Работа машины на основании заданной программы (и при заданном начальном состоянии) происходит следующим образом. Машина приводится в начальное состояние и приступает к выполнению первой команды программы (что значит «выполнить команду», будет объяснено ниже). Эта команда выполняется за один шаг, после чего машина приступает к выполнению той команды, номер которой (назовем его α) равен отсылке (одной из отсылок, если их две) первой команды. Эта команда также выполняется за один шаг, после чего начинается выполнение команды, номер которой равен отсылке команды с номером α. Вообще каждая команда выполняется за один шаг, а переход от выполнения одной команды к выполнению другой происходит по следующему правилу: пусть на k-м шаге выполнялась команда с номером i, тогда, если эта команда имеет единствен-

ную отсылку j, то на $k+1$-м шаге выполняется команда с номером j; если эта команда имеет две отсылки j_1 и j_2, то на $k+1$-м шаге выполняется одна из двух команд — с номером j_1 или с номером j_2 (какая именно, будет указано ниже); если, наконец, выполняющаяся на k-м шаге команда вовсе не имеет отсылки, то на $k+1$-м шаге и на всех последующих шагах не выполняется никакая команда: машина останавливается. Осталось объяснить, что значит выполнить команду и какая из отсылок — при наличии двух — выбирается в качестве номера следующей команды.

Выполнение команды движения вправо состоит в том, что каретка сдвигается на одну секцию вправо. Выполнение команды движения влево состоит в том, что каретка сдвигается на одну секцию влево. Выполнение команды печатания метки состоит в том, что каретка ставит метку на обозреваемой секции; выполнение этой команды возможно лишь в том случае, если обозреваемая перед началом выполнения команды секция пуста; если же на обозреваемой секции уже стоит метка, команда считается невыполнимой. Выполнение команды стирания метки состоит в том, что каретка уничтожает метку в обозреваемой секции; выполнение этой команды возможно лишь в том случае, если обозреваемая секция отмечена; если же на обозреваемой секции и так нет метки, команда считается невыполнимой. Выполнение команды передачи управления с верхней отсылкой j_1 и нижней отсылкой j_2 никак не изменяет состояния машины: ни одна из меток не уничтожается и не ставится, и каретка также остается неподвижной (машина делает, так сказать, «шаг на месте»); однако если секция, обозреваемая перед началом выполнения этой команды, была пуста, то следующей должна выполняться команда с номером j_1, если же эта секция была отмечена, следующей должна выполняться команда с номером j_2 (роль команды передачи управления сводится, следовательно, к тому, что каретка во время выполнения этой команды как бы «распознает», стоит ли перед ней метка,— именно это имелось в виду в предпоследней фразе § 1). Выполнение команды остановки тоже никак не меняет состояния машины и состоит в том, что машина останавливается.

Если теперь, задавшись программой и каким-либо начальным состоянием, пустить машину в ход, то осуществится один из следующих трех вариантов:

1) В ходе выполнения программы машина дойдет до выполнения невыполнимой команды (печатания метки в непустой секции или стирания метки в пустой секции); выполнение программы тогда прекращается, машина останавливается; происходит так называемая *безрезультатная остановка.*

2) В ходе выполнения программы машина дойдет до выполнения команды остановки; программа в этом случае считается выполненной, машина останавливается; происходит так называемая *результативная остановка.*

3) В ходе выполнения программы машина не дойдет до выполнения ни одной из команд, указанных в первых двух вариантах; выполнение программы при этом никогда не прекращается, машина никогда не останавливается; процесс работы машины происходит бесконечно.

§ 4. Примеры выполнения программ

Зададим, например, начальное состояние, указанное на рис. 5, и следующую программу:

1. V 4
2. § 3
3. ⇐2
4. ⇒5
5. ? 4 3

Посмотрим, как будет работать машина при таком начальном состоянии и такой программе.

На первом шаге будет выполняться команда № 1. После первого шага состояние машины станет таким,

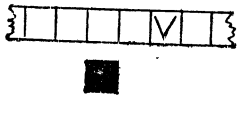


Рис. 5

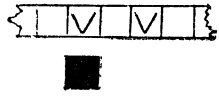


Рис. 6

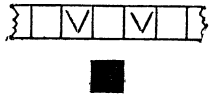


Рис. 7

как указано на рис. 6. После выполнения команды № 1 надо перейти к выполнению той команды, номер которой совпадает с отсылкой команды № 1, т. е. к выполнению команды № 4. Эта команда будет выполняться в течение второго шага, и состояние машины станет таким, как на рис. 7. Теперь надо выполнить команду № 5 (ибо отсылка команды № 4 равна 5). Эта команда будет выполняться на третьем шаге, в результате которого со-

стояние машины не изменится и останется таким, как на рис. 7. Поскольку обозреваемая секция при этом пуста, то следующей должна выполняться команда, номер которой равен верхней отсылке, т. е. числу 4. После выполнения на четвертом шаге команды № 4 машина придет в состояние, указанное на рис. 8. Теперь, на пятом шаге, будет выполняться команда № 5. На этот раз обозреваемая секция отмечена, поэтому следующей будет выполняться команда с номером, равным нижней отсылке, т. е. числу 3. После выполнения на шестом шаге

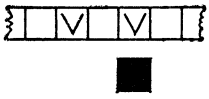


Рис. 8

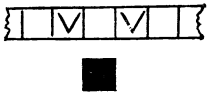


Рис. 9

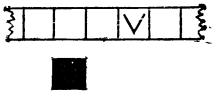


Рис. 10

команды № 3 машина придет в состояние, показанное на рис. 9, и приступит на седьмом шаге к выполнению команды № 2. Однако команда № 2 окажется невыполнимой, поскольку предписывает стереть метку в пустой секции. Следовательно, на седьмом шаге произойдет безрезультатная остановка.

Различные программы, примененные к одному и тому же начальному состоянию, могут приводить к различным исходам: безрезультатной остановке, результативной остановке, безостановочной работе машины. Действительно, зададимся, например, начальным состоянием, указанным на рис. 10. Применим к этому начальному состоянию программу

1. $\Rightarrow 2$
2. $\Rightarrow 3$
3. $\vee 1$.

Машина сделает два шага, а на третьем шаге произойдет безрезультатная остановка. Применим к этому начальному состоянию программу

1. $\Rightarrow 2$
2. $\Rightarrow 3$
3. стоп.

Машина сделает два шага, а затем на третьем произойдет результативная остановка. Применим, наконец,

к этому же начальному состоянию программу

1. ⇒1.

Машина будет работать бесконечно. Применим теперь программу

1. ? 1 1 .

Машина опять-таки будет работать бесконечно (несмотря на то, что ни запись на ленте, ни положение каретки не будут при этом меняться).

Точно так же различные варианты может давать и одна и та же программа, примененная к различным

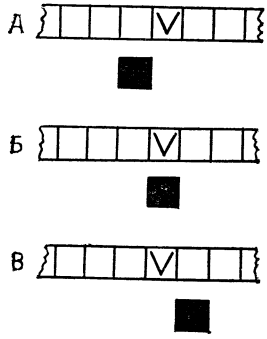


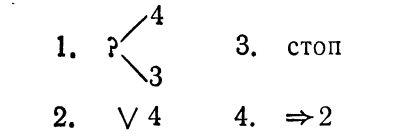
Рис. 11

начальным состояниям. Рассмотрим, например, следующую программу:

1. ? 4 1
2. §3
3. стоп
4. ⇒2.

Применим ее к начальным состояниям А, Б, В, показанным на рис. 11. Для начального состояния А получаем результативную остановку на четвертом шаге, для Б — бесконечную работу машины, для В — безрезультатную остановку на третьем шаге. Применение

к тем же начальным состояниям программы



дает безрезультатную остановку для А, результативную остановку для Б и бесконечную работу машины для В.

Упражнение 1. Может ли быть программа, дающая при любом начальном состоянии результативную остановку? Безрезультатную остановку? Неограниченное продолжение работы машины? Каково наименьшее число команд в этих программах?

Упражнение 2. О некоторой программе известно, что существует начальное состояние, для которого она дает результативную остановку, существует начальное состояние, для которого она дает безрезультатную остановку, и существует начальное состояние, для которого она дает непрекращающуюся работу машины. Докажите, что число команд в этой программе не менее 4. Напишите все такие программы длины 4.

§ 5. Методические замечания

Этот параграф адресован учителям начальной школы, руководителям школьных кружков для младших школьников и вообще всем тем, кто решит обучать младшеклассников машине Поста.

Как уже отмечалось в предисловии, работать с машиной Поста (изображенной при помощи рисунка мелом на доске или при помощи бумажной ленты и пуговиц или скрепок в качестве меток) можно обучить и первоклассников. Разумеется, для изложения младшим школьникам необходимо не только отказаться от некоторых деталей, но и ввести новые соглашения. Так, целесообразно:

1) ничего не говорить о системе координат на ленте (которая нужна лишь для уточнения того, что такое «состояние») и не вводить понятия «состояние» (для ленты или машины в целом);

2) в качестве номеров программ можно использовать не цифры (или не только цифры), но какие-либо изображения (например, изображения геометрических фигур);

3) предполагать ленту не бесконечной, а конечной, и договориться, что машина останавливается, когда каретка доходит до конца ленты;

4) о безрезультатной остановке вовсе ничего не говорить; если же она появится при исполнении какой-либо

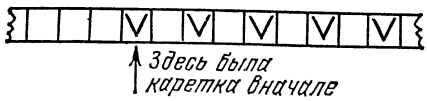


Рис. 12

программы, то говорить, что в этом случае «машина ломается»;

5) вводить команды не все сразу, а постепенно, сопровождая введение каждой новой команды наглядными упражнениями.

Упражнения, конечно, должны быть подобраны специально. Полезно предложить, например, такое

Упражнение. Машина начинает с «пустого» начального состояния (т. е. такого, при котором все секции пусты) и работает по программе

1. $\vee 2$
2. $\Rightarrow 3$
3. $\Rightarrow 1$;

спрашивается, что произойдет с лентой. Ответ достаточно нагляден; он показан на рис. 12.

Впрочем, мы считаем, что школьник должен (и может) уметь применять любую программу к любому начальному состоянию, так что любые, не требующие слишком большого времени упражнения на выполнение программ (и даже на составление некоторых программ, например, программы, приводящей к состоянию ленты, показанной на рис. 12) вполне уместны.

Разумеется, в изложении, предназначенном для младших школьников, не стоит употреблять слов «алгоритм», «машина Поста» (да и вообще «машина»; вместо слов «каретка передвигается на одну секцию вправо» лучше говорить, например, «мы переходим в соседнюю клетку справа» и т. д.). Более того, объяснение, зачем нужна излагаемая конструкция, целесообразно отложить до того времени, пока не будет достигнуто отчетливое ее понимание и достаточная непринужденность в выполнении программ. Здесь уместно привести следующую цитату

из эпиграфа к главе девятой книги: Сойер У. У. Прелюдия к математике: Пер. с англ.— М.: Просвещение, 1965: «Сначала я вам скажу, что я делаю, а потом объясню, зачем».

Вообще способность воспринимать какую-либо систему понятий или какое-либо построение до (и независимо от) получения информации о том, зачем это нужно, т. е. до (и независимо от) каких бы то ни было приложений, представляется нам одним из важнейших качеств, воспитываемых занятиями математикой. Представление о цели, которую преследует изложение того или иного материала, способствует, возможно, его запоминанию, но не должно влиять на понимание, которое может и должно иметь место независимо от этой цели. Умение мыслить формально — это особое умение, развивающееся, как и всякое умение, в результате тренировки. Такая тренировка могла бы начинаться с ранних лет. Доступными для первоклассника элементами такой тренировки могут служить и сложение многозначных чисел (при том, что многозначные числа понимаются без обычной количественной семантики, просто как цепочки цифр, а сумма определяется посредством алгоритма сложения столбиком[1])) и простейшие упражнения с машиной Поста.

Завершение изложения некоторой конструкции — изложения, непременно сопровождающегося достаточным числом примеров и упражнений,— означает конец важного этапа, вслед за которым, на следующем этапе, уже можно перейти к применению описанной конструкции. Для машины Поста таким применением служат вычисления, которые можно производить с ее помощью. Чтобы подчеркнуть рубеж, разделяющий эти два этапа, мы ограничиваемся в этой главе лишь первым знакомством с машиной Поста.

После такого знакомства уместно сообщить школьникам, что машина Поста может быть использована для получения результатов арифметических действий, и предложить примеры программ (а потом и задачи на составление программ), приводящих от записанных на ленте машины чисел к результатам тех или иных операций

[1]) См. Успенский В. А. К преподаванию математики в начальной школе//Математика в школе. — 1966. — № 2. — С. 38; имеется в виду, что лишь потом описанная процедура формального сложения используется для получения суммы двух количеств яблок или тетрадей.

над ними. Простейшей из таких операций является прибавление к числу единицы. Для этого, конечно, надо предварительно договориться, как записывать числа на ленте машины Поста, а также сделать еще ряд уточнений. Но это уже предмет следующей главы, которая будет специально посвящена прибавлению единицы. В качестве полезного упражнения можно предложить читателю самому попытаться вложить точный смысл в следующую формулировку: «Составить программу машины Поста, осуществляющую прибавление единицы». А может быть, таких смыслов окажется не один, а много?

ГЛАВА ВТОРАЯ

ПРИБАВЛЕНИЕ ЕДИНИЦЫ НА МАШИНЕ ПОСТА

На машине Поста можно совершать разнообразные вычисления. В настоящей главе вниманию читателя предлагается простейшее из таких вычислений — прибавление единицы к произвольному числу; это вычисление осуществляется в нескольких вариантах, зависящих от той или иной формулировки начальных условий.

Рассмотрение того, как на машине Поста производится прибавление единицы,— при всей элементарности этой темы — позволяет познакомить читателя (в очень упрощенной форме, конечно) с проблематикой, возникающей и при оперировании с реальными быстродействующими вычислительными машинами. Дело в том, что основная математическая задача, встающая перед человеком при работе его на вычислительных машинах, остается той же самой и для реальных и для «абстрактных» машин. Эта задача состоит в составлении для машины программы, приводящей к заданной цели: составление таких программ называется *программированием*. В данной главе рассматривается задача (точнее, серия задач) о составлении для машины Поста программ, приводящих к увеличению исходного числа на единицу.

Помимо общности основной задачи, и многие другие моменты программирования на машину (т. е. для машины) Поста характерны и для программирования на реальные машины. Вот некоторые из таких моментов, становящихся заметными уже при рассмотрении задачи о прибавлении единицы:

1. Можно составлять различные программы, приводящие к заданной (одной и той же) цели, т. е. осуществляющие заданную переработку информации;

читатель увидит (§ 2, упражнение), что для машины Поста возможно даже бесконечно много программ, осуществляющих прибавление единицы (уже при простейшем варианте такого прибавления).

2. Если расширяется класс исходных данных, в применении к которым программа должна приводить к нужному результату, то задача о построении такой программы становится более трудной, а сама программа, служащая решением задачи,— более сложной; читатель увидит, как будет усложняться программа прибавления единицы при расширении класса допустимых начальных условий.

3. При построении программ, дающих решение более общих или более сложных задач, часто бывает удобно использовать в качестве готовых строительных блоков составленные ранее программы для задач более частных или более простых; читатель увидит в § 4, как при построении программ для более общей задачи о прибавлении единицы можно использовать программы, дающие решение для более частных задач.

4. Составляя программу, приходится учитывать не только то, к каким числам ее надлежит применять, но и то, как эти числа расположены в «запоминающем устройстве», или «памяти», машины; читатель увидит, что разбираемые в этой статье варианты задачи о прибавлении единицы будут отличаться друг от друга лишь расположением исходного числа относительно каретки.

5. Стремление к тому, чтобы получающиеся программы были по возможности короткими, является вообще весьма естественным, а для реальных машин и задач в ряде случаев краткость программ может иметь решающее значение; читатель увидит, что на минимизацию длины программы будет обращено специальное внимание.

§ 1. Запись чисел в машине Поста и постановка задачи о прибавлении единицы

На машине Поста можно производить различные действия над числами. Для этого надо прежде всего договориться, как в машине Поста будут записываться числа. Речь будет всегда идти о целых неотрицательных числах 0, 1, 2, 3, 4, ...

Рассмотрим конечную последовательность идущих подряд друг за другом отмеченных секций ленты, заклю-

ченную между двумя пустыми секциями. Такую последовательность отмеченных секций будем называть *массивом*, а число секций в ней — *длиной* массива. Так, на рис. 13 показан массив длины 3, а на рис. 14 — три массива: длины 5, длины 1 и длины 2.

Условимся теперь число n записывать на ленте посредством массива длины $n+1$, а сам этот массив называть *машинной записью* числа n. На рис. 13 и 14 показаны, следовательно, машинные записи чисел 2, 4, 0 и 1.

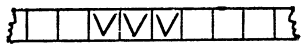


Рис. 13

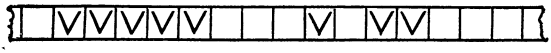


Рис. 14

Зададимся целью осуществить на машине Поста прибавление единицы. Понимать нашу задачу мы будем следующим образом: надо составить программу, которая, будучи применена к ленте, на которой записано произвольное число n, приводила бы к результативной остановке, причем после остановки на ленте должно быть записано число $n+1$ (имеется в виду составление одной программы, годящейся для любого n).

В предыдущей фразе задача еще поставлена не вполне точно, так как ни о начальном (т. е. задаваемом в начале), ни о заключительном (т. е. возникающем после результативной остановки) состоянии машины не сказано, ни где именно записано число, ни что еще записано на ленте; не сказано также, где должна стоять каретка. Будем предполагать, что в начале и в конце работы программы на ленте имеются только записи соответствующих чисел (n в начале и $n+1$ в конце), расположенные в произвольном месте ленты, а в остальном лента пуста. В целях облегчения нашей задачи не будем налагать никаких дополнительных ограничений на заключительное состояние: нас, таким образом, устроит любая программа, приводящая к ленте с записью числа $n+1$, где бы эта запись ни находилась и где бы ни стояла при этом каретка. В то же время мы будем делать все более и более широкие предположения

относительно взаимного расположения каретки и машинной записи числа в начальном состоянии машины. Таким образом, мы будем иметь дело не с одной задачей, а с целой серией задач. Мы настоятельно рекомендуем читателю, прежде чем читать решение какой-либо задачи, попытаться решить ее самому.

§ 2. Прибавление единицы в простейшем случае

Мы начнем с самого сильного ограничения, налагаемого на взаимное расположение машинной записи числа

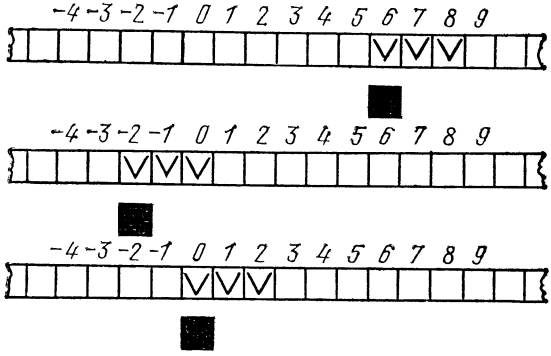


Рис. 15. Состояния из класса A_n отличаются друг от друга лишь «сдвигом» относительно системы координат

и каретки в начале, и, тем самым, с наиболее простой задачи. Назовем ее задачей 1.

Задача 1 (длинная формулировка). Требуется написать программу машины Поста, обладающую следующим свойством. Каково бы ни было число n, если начальное состояние машины Поста таково, что на ленте имеется машинная запись числа n (а в остальном лента пуста) и каретка стоит против самой левой секции записи, то выполнение программы должно привести к результативной остановке, после чего на ленте (в произвольном ее месте) должно быть записано число $n + 1$ (а в остальном лента должна быть пуста), причем каретка может стоять где угодно.

Обозначим через A_n совокупность всех таких состояний машины Поста, в каждом из которых отмеченными на ленте являются равно $n + 1$ секций, а каретка стоит против самой левой из отмеченных секций. На рис. 15 показано несколько состояний из класса A_2; они отли-

чаются друг от друга лишь различным положением массива и «жестко связанной с ним» каретки относительно начала координат.

Обозначим через E_n совокупность всех таких состояний машины Поста, в каждом из которых отмеченными на ленте являются ровно $n+1$ секций, а каретка может стоять где угодно. К E_n относятся все состояния из A_n

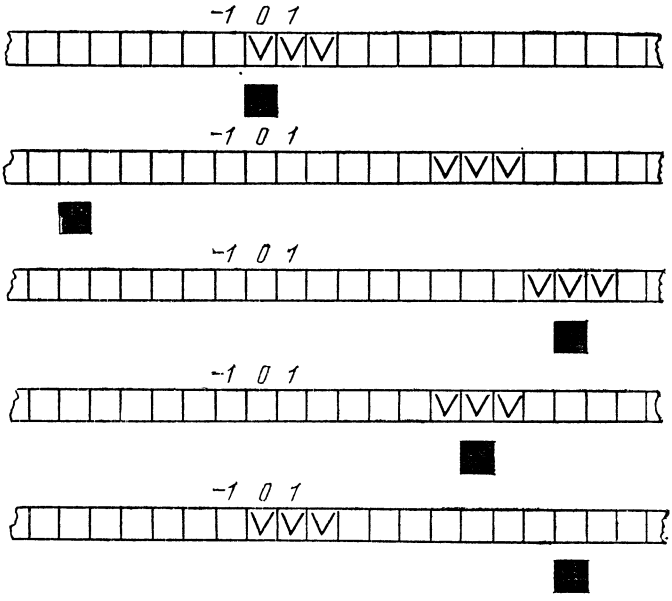


Рис. 16

и еще много других. На рис. 16 показано несколько состояний из класса E_2.

Теперь задачу 1 можно сформулировать короче.

Задача 1 (короткая формулировка). Написать такую программу машины Поста, которая для любого n, будучи применена к произвольному состоянию из класса A_n, дает результативную остановку в каком-то состоянии из класса E_{n+1}.

Прежде чем приступить к решению задачи 1, заметим, что в ее условии ничего не говорится о том, что должно получаться при применении искомой программы к состояниям, не принадлежащим ни к одному из классов A_n ($n=0, 1, 2, \ldots$); это значит, что нам безразлично, что будет происходить при таких состояниях,

выбираемых в качестве начальных: нас устроит любая программа, переводящая состояния из A_n в состояния из E_{n+1}, что бы она ни делала с остальными состояниями.

Решением задачи 1 будет, например, такая программа:

Программа I_1

1. $\Leftarrow 2$
2. $\vee 3$
3. стоп.

Мы написали «например», потому что решение задачи 1 не единственно: возможны и другие программы, удовлетворяющие условиям задачи. Например, такая программа также будет решением задачи 1:

1. $\Rightarrow 2$
2. ? → 3, 3
3. $\Leftarrow 4$
4. $\Leftarrow 5$
5. $\vee 6$
6. стоп.

Однако написанная выше программа I_1 будет самой короткой (точнее, одной из самых коротких) из программ, удовлетворяющих условиям задачи 1. Действительно, можно доказать (и мы рекомендуем это сделать читателю), что никакая программа длины 1 или 2 уже не может служить решением задачи 1; в то же время есть еще ровно одна программа длины 3, также являющаяся решением нашей задачи; вот эта программа:

Программа I_2

1. $\Leftarrow 3$
2. стоп
3. $\vee 2$.

Упражнение. Докажите, что существует бесконечно много программ, являющихся решениями задачи 1.

Замечание. При фиксированном n состояния из A_n отличаются друг от друга лишь сдвигом относительно начала координат (рис. 15). Какую бы программу мы ни применили к этим начальным состояниям, получающиеся результаты будут, очевидно, отличаться друг от друга тем же самым сдвигом. Поэтому достаточно выбрать из каждого класса A_n по одному состоянию a_n и составить программу, которая каждое a_n будет переводить в некоторое состояние из E_{n+1}. Всякая такая программа автоматически будет решением задачи 1. Заметим, что состояние a_n можно выбирать из A_n совершенно произвольно.

Совершенно аналогично задаче 1 может быть решена задача 1′, отличающаяся от задачи 1 лишь тем, что вначале каретка обозревает самую правую из секций массива. Именно, обозначим через A'_n класс всех таких состояний из E_n, в которых каретка обозревает самую правую из отмеченных секций. Тогда короткая формулировка задачи 1′ будет такова:

Задача 1′ (короткая формулировка). Написать такую программу машины Поста, которая для любого n, будучи применена к произвольному состоянию из класса A'_n, дает результативную остановку в каком-то состоянии из класса E_{n+1}.

Следующие две программы будут единственными самыми короткими программами, удовлетворяющими условию задачи 1′:

Программа I'_1

1. $\Rightarrow 2$
2. $\vee 3$
3. стоп.

Программа I'_2

1. $\Rightarrow 3$
2. стоп
3. $\vee 2$.

§ 3. Прибавление единицы в более сложных случаях

Не будем теперь требовать, чтобы каретка в начале стояла непременно против одной из крайних секций массива, как это было в задачах 1 и 1′. Ограничимся требованием, чтобы в начальном состоянии каретка обозревала одну из секций массива.

Задача 2 (длинная формулировка). Требуется написать программу машины Поста, обладающую следующим свойством. Каково бы ни было число n, если начальное состояние машины Поста таково, что на ленте имеется машинная запись числа n (а в остальном лента пуста) и каретка стоит против одной из секций записи, то выполнение программы должно привести к результативной остановке, после чего на ленте (в произвольном ее месте) должно быть записано число $n+1$

(а в остальном лента должна быть пуста), причем каретка может стоять где угодно.

Обозначим через B_n совокупность всех таких состояний машины Поста, в каждом из которых отмеченными на ленте являются ровно $n+1$ секций, а каретка стоит против одной из отмеченных секций. Очевидно, класс B_n является частью класса E_n и сам содержит в качестве

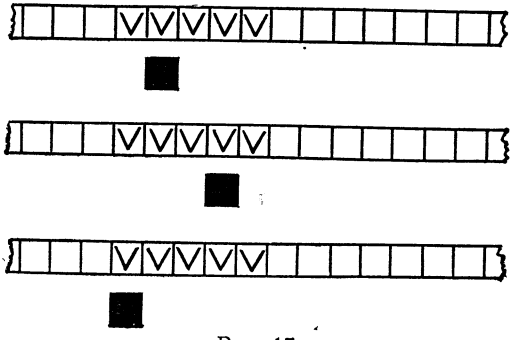


Рис. 17

Рис. 18. Общий вид состояния из класса B_n. Здесь $n' \geqslant 0$, $n'' > 0$, $n' + n'' = n$

части класс A_n. На рис. 17 изображено несколько состояний из класса B_4, а на рис. 18 — общий вид состояния из класса B_n. Тогда получаем следующую короткую формулировку задачи 2:

Задача 2 (короткая формулировка). Написать такую программу машины Поста, которая для любого n, будучи применена к произвольному состоянию из класса B_n, дает результативную остановку в каком-то состоянии из класса E_{n+1}.

Очевидно, что каждое решение задачи 2 будет одновременно и решением задачи 1, а также задачи 1′. В то же время существуют решения задачи 1, не являющиеся решениями задачи 2. Таковы, например, программы I_1 и I_2. Так же, как и для задачи 1, для задачи 2 существует бесконечное множество программ, являющихся ее решением. Мы по-прежнему будем интересоваться наибо-

лее короткими программами. Можно доказать (рекомендуем читателю это сделать), что никакая программа длины 1, 2 или 3 не может быть решением задачи 2. В то же время возможны решения задачи 2 длины 4. Вот одно из таких решений:

Программа $П_1$

1. $\Leftarrow 2$ 3. $\vee 4$

2. $?\begin{matrix}\nearrow 3\\ \searrow 1\end{matrix}$ 4. стоп.

Упражнение 1. Найдите еще два решения задачи 2, имеющих длину 4 и содержащих команду движения влево. Проверьте, что каждое из найденных вами решений содержит команду печатания метки, команду передачи управления и команду остановки.

Упражнение 2. Докажите, что, помимо программы $П_1$, существует еще ровно одиннадцать ($П_2$, $П_3$, ..., ..., $П_{12}$) решений задачи 2, имеющих длину 4 и содержащих команду движения влево. Выпишите эти решения.

Упражнение 3. Проверьте, что программы $П'_1$, $П'_2$, ..., $П'_{12}$, получающиеся из программ $П_1$, $П_2$, ..., $П_{12}$ путем замены знака $\Leftarrow$ на знак $\Rightarrow$, также служат решениями задачи 2. Докажите, что программами $П_1$, $П_2$, ..., $П_{12}$, $П'_1$, $П'_2$, ..., $П'_{12}$ исчерпываются все решения задачи 2 длины 4.

Рассмотрим теперь такие начальные состояния, в которых каретка держит в поле зрения не какую-то из секций исходного массива, а какую-то пустую секцию. Будем предполагать при этом, что является заранее известным, стоит ли каретка слева или справа от исходного массива. Слова «является заранее известным» означают, что требуется найти не единую программу, работающую во всех случаях, а две программы, одна из которых работает в случае, если каретка вначале стоит слева от массива, а другая работает в случае, когда каретка вначале стоит справа от массива. Таким образом, мы имеем здесь не одну, а две задачи. Чтобы получить сразу короткие формулировки этих задач, введем следующие обозначения. Обозначим через C_n совокупность всех таких состояний из класса E_n, в которых каретка стоит слева от массива, а через C'_n — совокупность всех

таких состояний из класса E_n, в которых каретка стоит справа от массива. На рис. 19 показан общий вид состояния из класса C_n, а на рис. 20 — общий вид состояния из класса C'_n.

Задача 3 (короткая формулировка). Написать такую программу машины Поста, которая для любого n, будучи применена к произвольному состоянию

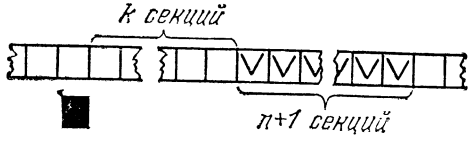


Рис. 19. Общий вид состояния из класса C_n. Здесь $k \geqslant 0$

Рис. 20. Общий вид состояния из класса C'_n. Здесь $k \geqslant 0$

из класса C_n, дает результативную остановку в каком-то состоянии из класса E_{n+1}.

Задача 3′ (короткая формулировка). Написать такую программу машины Поста, которая для любого n, будучи применена к произвольному состоянию из класса C'_n, дает результативную остановку в каком-то состоянии из класса E_{n+1}.

Очевидно, что каждую программу, являющуюся решением задачи 3, можно превратить в решение задачи 3′, если всюду знак $\Rightarrow$ заменить на знак $\Leftarrow$, а знак $\Leftarrow$ заменить на знак $\Rightarrow$. Такой же операцией любое решение задачи 3′ превращается в решение задачи 3. Поэтому достаточно решить только одну из этих задач. Вот одно из решений задачи 3:

Программа III

1. $\Rightarrow 2$
2. ? 1 / 3
3. $\Leftarrow 4$
4. $\vee 5$
5. стоп.

Попробуйте доказать, что задача 3 не допускает более коротких решений. Сколько существует решений этой задачи длины 5?

§ 4. Прибавление единицы в еще более сложном случае

В этом параграфе мы рассмотрим в качестве класса исходных (начальных) состояний объединение всех классов B_0, C_0, B_1, C_1, B_2, C_2, B_3, C_3, ... из предыдущего параграфа. Этот класс будет, следовательно, состоять из всех таких — и только таких — состояний машины, в каждом из которых каретка стоит либо против какой-то из секций исходного массива, либо же левее массива.

Обозначим через D_n совокупность всех таких состояний из класса E_n, в которых обозреваемая кареткой секция либо отмечена, либо расположена левее всех отмеченных секций. Очевидно, что при каждом n класс D_n есть объединение классов B_n и C_n.

Задача 4. Написать такую программу машины Поста, которая для любого n, будучи применена к произвольному состоянию из класса D_n, дает результативную остановку в каком-то состоянии из класса E_{n+1}.

Задачу 4 можно свести к задачам 2 и 3, т. е. можно указать способ, как из произвольных решений задач 2 и 3 получить некоторое решение задачи 4. Укажем такой способ. Пусть Ξ — произвольный список команд машины Поста, а k — произвольное целое число. Через $\Xi[+k]$ будем обозначать список, полученный из Ξ путем прибавления числа k ко всем номерам и всем отсылкам команд из Ξ. Например, $\mathrm{I}_1[+7]$ — это такой список:

8. $\Leftarrow 9$
9. $\vee\, 10$
10. стоп.

Пусть теперь II — произвольная программа, являющаяся решением задачи 2, а III — произвольная программа, являющаяся решением задачи 3. Пусть l есть длина программы II. Составим теперь такой список команд:

1. ? $\begin{cases} l+2 & \text{II } [+1] \\ 2 & \text{III } [+l+1] \end{cases}$

Легко проверить, что этот список команд является программой машины Поста, причем такой программой, которая удовлетворяет условиям задачи 4. В самом деле, всякое состояние из класса D_n принадлежит либо B_n, либо C_n. В первом случае «срабатывает» список II[+1], во втором — список III[+l + 1].

Однако изложенный способ не обязан давать (и, как мы сейчас увидим, действительно не дает) кратчайшее решение задачи 4 — даже если исходить из кратчайших решений задач 2 и 3. Посмотрим, что получится, если применить этот способ к программам II_1 и III. Мы получим такое решение задачи 4:

Программа IV[10]

1. ? 6 / 2
2. ⇐ 3
3. ? 4 / 2
4. ∨ 5
5. стоп
6. ⇒ 7
7. ? 6 / 8
8. ⇐ 9
9. ∨ 10
10. стоп.

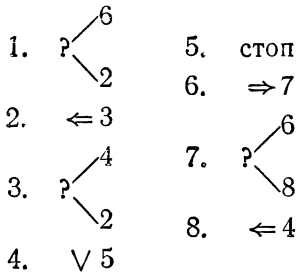
Ясно, что эту программу можно сократить, не меняя при этом совершаемой ею работы, путем объединения в одну двух команд остановки, или, как мы будем говорить, путем поглощения одной из этих команд другой. Проще поглотить команду № 10 командой № 5. Для этого достаточно заменить отсылку 10 во всех командах, где она встречается, — в нашем случае только в команде № 9 — на отсылку 5 и затем вычеркнуть команду № 10 [1]).

Мы получим программу IV[9], в которой можно теперь поглотить команду № 9 командой № 4. Для этого достаточно заменить в команде № 8 отсылку 9 на отсылку 4 и затем вычеркнуть команду № 9. После произведенных двух поглощений получим новое решение задачи 4 в виде программы IV[8].

---

[1]) Если бы мы хотели поглотить команду № 5 командой № 10, то надо было бы заменить отсылку 5 на отсылку 10 во всех командах, где она встречается (в данном случае в команде № 4), затем вычеркнуть команду № 5, после чего уменьшить на 1 все номера команд, следующих за вычеркиваемой, и все отсылки, совпадающие с этими номерами.

Программа IV[8]



Замечание. В общем случае поглощение (в данной программе) команды № α командой № β состоит в последовательном выполнении трех операций. Во-первых, всюду отсылка α заменяется на отсылку β; во-вторых, команда № α вычеркивается; в-третьих, во всех командах образовавшегося списка числа $\alpha+1$, $\alpha+2$, $\alpha+3$, ... (которые могут быть как номерами, так и отсылками) заменяются соответственно на α, $\alpha+1$, $\alpha+2$, ... Если команды № α и № β совпадали во всем, кроме своих номеров, то программа, образовавшаяся после поглощения команды α командой β, и первоначальная программа будут «работать совершенно одинаково». Слова, взятые в предыдущей фразе в кавычки, уточняются следующим образом. Возьмем два экземпляра машины Поста, приведем каждый из них в некоторое — одно и то же для обеих машин — начальное состояние и заставим работать первую машину по некоторой программе А, а вторую — по некоторой программе Б. Предполагая, что машины работают синхронно, будем сравнивать одновременно возникающие состояния первой и второй машины. В начальный момент эти состояния совпадают по условию. Может случиться, что они совпадают и во все последующие моменты, причем остановка каждой машины происходит — если только происходит вообще — одновременно с остановкой другой машины и имеет то же качество (т. е. эта остановка либо у обеих машин результативная, либо у обеих машин безрезультатная). Если описанное явление осуществляется для любого начального состояния, общего для обеих машин, то мы и будем говорить, что программы А и Б работают совершенно одинаково. Более грубо, А и Б работают совершенно одинаково, коль скоро для любого m

по прошествии m шагов работы программы А возникает то же состояние, что после m шагов работы программы Б, — при условии, что это было верно для $m = 0$, т. е. в самом начале работы. Вот примеры одинаково работающих программ: а) I_1 и I_2; б) I_1' и I_2'; в) IV^{10} и IV^8.

Посмотрим теперь, нельзя ли сократить и программу IV^8. В ней нет двух команд, различающихся только номерами, поэтому способ поглощения здесь может, вообще говоря, привести к программе, которая не будет работать совершенно одинаково с исходной. Все же и программу IV^8 можно сократить способом поглощения.

С этой целью сравним команды № 8 и № 2. Их нельзя объединить в одну, поскольку они имеют разные отсылки. Оказывается, тем не менее, что команда № 2 может, в известном смысле, взять на себя функции команды № 8 (и потому поглотить последнюю).

В самом деле, пусть на некотором этапе работы программы нам предстоит выполнять команду № 8. Предположим, что секция, расположенная непосредственно левее той, которая обозревается в рассматриваемый момент кареткой, пуста. Тогда команда № 8 сдвинет каретку в эту пустую секцию, после чего команда № 4 поставит на ней метку. Если вместо команды № 8 мы станем выполнять команду № 2, то произойдет вот что: команда № 2 сдвинет каретку в ту же, что и раньше, пустую секцию, команда № 3 заставит машину сделать «шаг на месте», после чего опять-таки сработает команда № 4. Посмотрим, что случится, если секция, расположенная непосредственно левее обозреваемой, не пуста, а отмечена. Если так, то в первом случае (т. е. после выполнения команды № 8) произойдет безрезультатная остановка, а во втором случае (т. е. после выполнения команды № 2) таковой не произойдет.

Проведенное рассуждение показывает, что если последовательное выполнение команд № 8 и № 4 не приводит к безрезультатной остановке, то оно может быть заменено выполнением команд №№2, 3, 4 (в то время как обратная замена может, вообще говоря, существенно изменить работу программы: выполнение команд №№ 2, 3, 4 никогда не приводит к безрезультатной остановке, но при замене этого выполнения выполнением команд №№ 8, 4 может возникнуть безрезультатная остановка). Поэтому, если ограничиться такими начальными состояниями, для которых при выполнении программы IV^8 невозможна безрезультатная остановка (а к таким заве-

домо относятся все состояния из D_n, где $n = 0, 1, 2, \ldots$,[1])), то программа IV^7, получающаяся после поглощения команды № 8 командой № 2, также будет решением задачи 4. Вот эта программа:

Программа IV^7

1.	$?\begin{cases}6\\2\end{cases}$	4.	$\vee 5$
		5.	стоп
2.	$\Leftarrow 3$	6.	$\Rightarrow 7$
3.	$?\begin{cases}4\\2\end{cases}$	7.	$?\begin{cases}6\\2.\end{cases}$

Упражнение. Будут ли программы IV^8 и IV^7 работать совершенно одинаково?

Произведя в программе IV^7 поглощение команды № 7 командой № 1, получим следующую программу — IV^6:

Программа IV^6

1.	$?\begin{cases}6\\2\end{cases}$	4.	$\vee 5$
2.	$\Leftarrow 3$	5.	стоп
3.	$?\begin{cases}4\\2\end{cases}$	6.	$\Rightarrow 1.$

Поскольку программы IV^7 и IV^6 работают совершенно одинаково, то программа IV^6, так же как и IV^7, будет решением задачи 4. Попробуйте доказать, что задача 4 не имеет более коротких решений.

Совершенно аналогично решается задача о прибавлении единицы для начальных состояний, в которых обозреваемая секция либо отмечена, либо стоит правее отмеченного массива. Решение соответствующей задачи — назовем ее задачей 4′ — можно получить, заменив в произвольном решении задачи 4 все знаки $\Rightarrow$ на $\Leftarrow$, а знаки $\Leftarrow$ на $\Rightarrow$.

---

[1]) Ведь в применении к этим состояниям программа IV^8 приводит к результативной остановке,

Если обозначить через D'_n совокупность всех таких состояний из E_n, в которых каретка стоит либо на массиве, либо правее массива, то можно заметить, что — при каждом n — класс E_n есть объединение классов. D_n и D'_n, а класс B_n — пересечение этих классов. В символах:

$$D_n \cup D'_n = E_n, \quad D_n \cap D'_n = B_n.$$

§ 5. Прибавление единицы в самом общем случае

Не будем теперь налагать никаких ограничений на взаимное расположение каретки и машинной записи числа в начальном состоянии. Требуется, следовательно, составить программу, которая осуществляла бы прибавление единицы при условии, что запись исходного числа находится в произвольном участке ленты и каретка стоит где угодно. Соответствующее требование формулируется в задаче 5.

Задача 5. Написать такую программу машины Поста, которая для любого n, будучи применена к произвольному состоянию из класса E_n, дает результативную остановку в каком-то состоянии из класса E_{n+1}.

Очевидно, что каждая программа, являющаяся решением задачи 5, будет одновременно служить решением каждой из предшествующих задач. Однако ни одно из приведенных выше решений задач 1, 1′, 2, 3, 3′, 4, 4′ не является решением задачи 5 (проверьте это).

Начальные состояния для задачи 5 показаны на рис. 18, 19 и 20: ведь любое состояние из E_n принадлежит либо B_n, либо C_n, либо C'_n. Однако теперь нужно составить единую программу, приводящую к цели для каждого из видов начального состояния, показанных на рис. 18, 19, 20. Попытайтесь найти такую программу самостоятельно. Вы увидите, что это не так просто. А может быть, требуемой программы не существует вовсе? Тогда попытайтесь доказать, что ее не существует. Ответ на вопрос, имеет ли задача 5 решение, будет дан в следующей главе.

ГЛАВА ТРЕТЬЯ

АНАЛИЗ И СИНТЕЗ ПРОГРАММ МАШИНЫ ПОСТА

В предшествующих главах мы уже встречались как с задачей синтеза программ (состоящей в построении программ, осуществляющих заданные действия), так и с задачей анализа (состоящей в описании тех преобразований, которые совершают программы). В данной главе эти задачи будут решаться для более сложных и содержательных ситуаций: анализ программ будет рассматриваться в связи с одной задачей, оставшейся нерешенной в предыдущей главе, а синтез — в связи с задачей о сложении чисел.

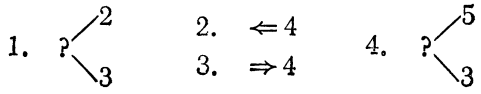
§ 1. Диаграммы и блок-схемы

Программы, с которыми мы встречались в предшествующих двух главах, посвященных машине Поста, были простыми.

Поэтому мы без труда могли как проанализировать заданную программу, т. е. понять, как она работает, так и синтезировать нужную программу, т. е. построить программу с требуемыми свойствами.

Но предположим, что дана программа посложнее, например, такая:

Программа V



5.	$\vee$ 6	11.	$\Rightarrow$ 12	17.	? 23, 18
6.	$\Leftarrow$ 7	12.	? 13, 19	18.	ξ 16
7.	? 8, 15	13.	$\Leftarrow$ 14	19.	$\Leftarrow$ 20
8.	$\Rightarrow$ 9	14.	? 5, 13	20.	$\Leftarrow$ 21
9.	? 10, 8	15.	$\Rightarrow$ 16	21.	? 23, 22
10.	$\vee$ 11	16.	$\Rightarrow$ 17	22.	ξ 20
				23.	стоп.

Как проанализировать эту программу?

Мы поступим так: разобьем нашу программу на части (которые назовем *блоками*) и постараемся понять, как работает каждый блок в отдельности и как отдельные блоки взаимодействуют между собой. Чтобы найти удобное разбиение программы V на блоки, составим так называемую *диаграмму* этой программы.

На диаграмме программы команды изображаются кружками, а переходы от одной команды к другой — стрелками. Чтобы построить диаграмму программы V, нарисуем 23 кружка и пометим их числами от 1 до 23. Проведем от кружка № i к кружку № j стрелку в том и только в том случае, если i-я команда (т. е. команда с номером i) имеет отсылку, равную j. В частности, если i-я команда есть команда остановки, то от i-го кружка не будет отходить ни одной стрелки, а если i-я команда есть команда передачи управления, то от i-го кружка будут отходить две стрелки — одна к кружку, отвечающему верхней отсылке команды, и вторая к кружку, отвечающему нижней отсылке. Вся наша программа изобразится тогда фигурой, показанной на рис. 21. Эту фигуру и назовем *диаграммой* программы V, а составляющие ее кружки — *узлами* диаграммы.

Изложенным способом можно построить диаграмму для любой программы. Последовательная смена выполняемых команд данной программы наглядно иллюстрируется движением по диаграмме этой программы в направлении стрелок.

Разобьем теперь нашу программу V на группы команд — блоки. Тем самым на блоки (группы узлов)

разобьется и диаграмма. Нагляднее задавать разбиение на блоки именно как разбиение диаграммы.

Для наших целей — целей анализа программы V — удобно разбить диаграмму рис. 21, а вместе с нею и саму программу V на следующие восемь блоков:

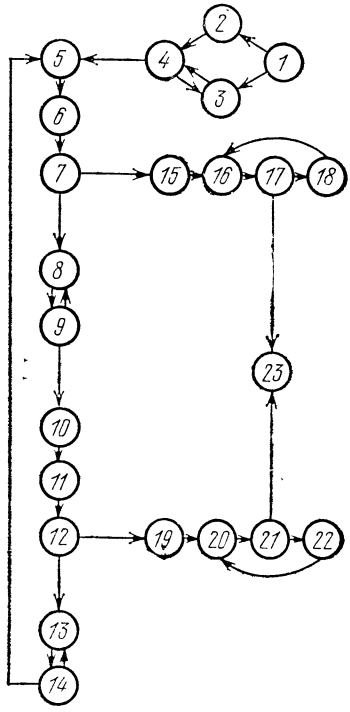


Рис. 21

1-й блок — назовем его «блоком начала» — состоит из узлов (команд) № 1, 2, 3, 4;

2-й блок — назовем его «блоком проверки влево» — состоит из узлов (команд) № 5, 6, 7;

3-й блок — назовем его «блоком движения вправо» — состоит из узлов (команд) № 8, 9;

4-й блок — назовем его «блоком проверки вправо» — состоит из узлов (команд) № 10, 11, 12;

5-й блок — назовем его «блоком движения влево» — состоит из узлов (команд) № 13, 14;

6-й блок — назовем его «блоком стирания вправо» — состоит из узлов (команд) № 15, 16, 17, 18;

7-й блок — назовем его «блоком стирания влево» — состоит из узлов (команд) № 19, 20, 21, 22;

8-й блок — назовем его «блоком остановки» — состоит из узла (команды) № 23.

Для каждого разбиения данной программы на блоки можно составить свою так называемую *блок-схему* данной программы. Для этого надо каждый блок изобразить в виде прямоугольника и провести от прямоугольника, изображающего блок α, к прямоугольнику, изображающему блок β, столько стрелок, сколько их идет от узлов блока α к узлам блока β. Блок-схема программы V для выбранного нами разбиения на блоки приведены на рис. 22 (внутри каждого прямоугольника вписан номер и название соответствующего блока).

Имея перед глазами блок-схему какой-либо программы, можно наглядно представить себе ее работу в форме последовательного выполнения команд то одного блока, то другого. Блок, содержащий команду № 1, будем называть *начальным*, а блок, содержащий команду остановки, — *заключительным*; для простоты будем считать, что заключительный блок ничего больше не содержит. Вместо того чтобы говорить «выполнение команд данного блока», условимся для краткости говорить просто «выполнение данного блока». Первым всегда будет выполняться начальный блок.

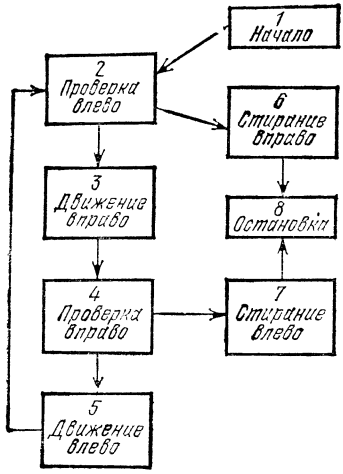


Рис. 22

При выполнении какого-либо незаключительного блока могут произойти три случая: 1) во время выполнения блока произойдет безрезультатная остановка; 2) выполнение блока никогда не окончится, т. е. каждый раз после выполнения какой-то команды надо будет снова выполнять команду того же блока; 3) выполнение блока окончится, т. е. после выполнения некоторой команды надо будет переходить к выполнению некоторой команды другого блока. Короче, попав в незаключительный блок, мы либо «застрянем» в нем навсегда, либо по одной из отходящих от него стрелок выйдем в некоторый другой блок.

Иногда можно сразу, по виду диаграммы, выделить некоторые блоки, в которых заведомо невозможно застрять (если только исключить случай безрезультатной остановки, которая может произойти в любой момент). Для блок-схемы на рис. 22 такими будут, как следует из диаграммы на рис. 21, блоки 2-й и 4-й. Поэтому выполнение программы V будет происходить следующим образом: сперва выполняется 1-й блок; его выполнение либо никогда не кончается, либо кончается, и тогда выполняется 2-й блок. Выполнение 2-го блока кончается обязательно, и после него выполняется либо 3-й блок, либо 6-й; после 3-го (если его выполнение заканчивается) — 4-й; после 4-го — либо 5-й, либо 7-й; после 5-го

(если его выполнение заканчивается) — 2-й; после 2-го снова либо 3-й, либо 6-й; и так далее. Выполнение 6-го (равно как и 7-го) блоков может закончиться, а может и не закончиться; если оно закончится, то выполняется 8-й блок, и машина останавливается.

Теперь мы можем, наконец, решить для нашей программы задачу анализа. Мы ограничимся при этом случаем, когда начальное состояние принадлежит (при некотором n) классу E_n. При помощи блок-схемы в следующем параграфе будет показано, что в применении к такому начальному состоянию программа приводит к результативной остановке в состоянии из класса E_{n+1}.

Тем самым решается следующая (поставленная в § 5 предыдущей главы) общая задача о прибавлении единицы:

написать программу машины Поста, которая для любого n, будучи применена к произвольному состоянию из E_n (взятому в качестве начального), дает результативную остановку в каком-то состоянии из класса E_{n+1}.

Именно, программа V длины 23 оказывается решением этой задачи. Автору не удалось построить более короткую программу, которая бы служила решением той же задачи. В то же время автор не знает, как доказать, что найденная программа — самая короткая из возможных. Может быть, кому-нибудь из читателей удастся сделать то или другое?

§ 2. Анализ программы прибавления единицы

Итак, перейдем к анализу программы V из § 1, т. е. к выяснению того, как она работает. Как мы договорились, в качестве начального состояния мы выбираем какое-то состояние, принадлежащее какому-то из классов $E_n (n = 0, 1, 2, \ldots)$. Обозначим через H_n класс тех состояний из класса E_n, при котором как обозреваемая секция, так и соседняя с нею секция справа обе пусты.

Приступаем к выполнению программы V. Сначала выполняется первый блок. Сейчас мы покажем, что выполнение его закончится (т. е. мы выйдем из него во второй блок); одновременно мы увидим, какое состояние машины возникнет после его выполнения. Рассмотрим три случая.

Случай 1. В начальном состоянии как обозреваемая секция, так и соседняя с ней слева секция обе пусты. В этом случае последовательно сработают команды № 1, 2, 4, после чего выполнение блока окончится, причем машина придет в состояние из класса H_n.

Случай 2. В начальном состоянии обозреваемая секция пуста, но соседняя с ней слева секция отмечена. Это значит, что интересующий нас массив длины $n + 1$ находится слева от обозреваемой секции и, следовательно, справа от нее все секции пусты.

В этом случае последовательно сработают команды № 1, 2, 4, 3, 4, после чего выполнение блока окончится, причем машина придет в состояние из класса H_n.

Случай 3. В начальном состоянии обозреваемая секция отмечена. Это значит, что каретка стоит как раз против одной из секций нашего масива длины $n+1$; пусть она стоит на k-й справа секции этого массива. В этом случае будут работать команды № 1, 3, 4, 3, 4, 3, 4, ..., причем выполнение пары команд № 3 и 4 произойдет ровно k раз, пока каретка не окажется на первой справа пустой секции; после этого выполнение блока закончится, и машина будет находиться в одном из состояний класса H_n.

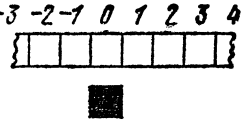


Рис. 23

Итак, к какому бы состоянию из класса E_n ни применялась программа V, выполнение 1-го блока закончится, после чего машина будет находиться в одном из состояний класса H_n.

Поэтому для осуществления нашей цели (состоящей в установлении того, что программа V является решением общей задачи о прибавлении единицы) достаточно доказать, что, применяя эту программу, *начиная со 2-го блока*, к произвольному состоянию из класса H_n, мы получим рано или поздно результативную остановку в состоянии из класса E_{n+1}. К доказательству этого факта мы сейчас и приступим.

Итак, пусть машина находится в некотором состоянии из класса H_n. Фиксируем это состояние — обозначим его h. Мы знаем, что на ленте введена «целочисленная система координат», согласно которой секции ленты занумерованы целыми числами. Забудем эту «старую» систему координат и введем следующую новую (также целочисленную) систему: обозреваемую в состоянии h секцию занумеруем числом 0; секции, стоящие справа и слева от нее, — числами ±1, ±2, ±3, ... (рис. 23). Подчеркнем, что эта система координат зависит от рассматриваемого нами состояния h из H_n. Тогда массив, служащий записью числа n, располагается в секциях с номерами m, $m+1$, ..., $m+n$, где m — какое-то число, положительное или отрицательное. Секции с номерами 0 и 1 заведомо пусты. Поэтому либо $m+n<0$, либо $m<1$.

Обозначим через H_n^m такое состояние машины, при котором отмечены секции с номерами m, $m+1$, ..., $m+n$, и только они, а каретка стоит против секции № 0. Тогда класс состояний H_n состоит из состояний H_n^m, где либо $m+n<0$, либо $m>1$, т. е. из состояний

а) H_n^{-n-1}, H_n^{-n-2}, H_n^{-n-3}, ...;

б) H_n^2, H_n^3, H_n^4, ...

Общий вид этих состояний показан на рис. 24. Наше исходное состояние h есть одно из этих состояний H_n^m.

Стало быть, нам нужно доказать, что под действием программы, начиная со 2-го блока, каждое из состояний H_n^m, где либо $m+n<0$, либо $m>1$, переходит с результативной остановкой в состояние из класса E_{n+1}. Это мы сейчас и установим путем наблюдения того, какие изменения претерпевает состояние H_n^m при последовательном выполнении блоков программы.

С этой целью введем некоторые обозначения. Пусть целые числа m, n, p, q удовлетворяют следующим трем условиям: 1) $p \leqslant q$; 2) $n \geqslant 0$; 3) либо $m+n<p$, либо $m>q$. Через $W_n^m(p, q)$ обозначим класс всех состояний машины Поста, которые удовлетворяют

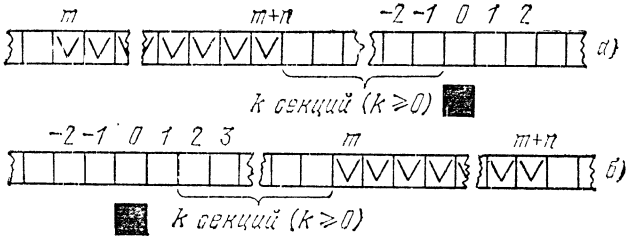


Рис. 24. Состояние H_n^m: *а*) при $m+n<0$; *б*) при $m>1$

следующим двум условиям: 1) секции с номерами p и q пусты, а все секции между ними отмечены; 2) секции с номерами $(m-1)$ и $(m+n+1)$ пусты, а все секции между ними отмечены.

Общий вид состояний класса $W_n^m(p, q)$ показана на рис. 25. Через $R_n^m(p, q)$ обозначим то состояние из класса $W_n^m(p, q)$, в котором каретка стоит против секции с номером p, а через $S_n^m(p, q)$ — то состояние из класса $W_n^m(p, q)$, в котором каретка стоит против секции с номером q.

Непосредственной проверкой можно установить следующие утверждения:

1°. $H_n^m = R_n^m(0, 1)$.

2°. Если перед началом работы второго блока состояние машины Поста было $R_n^m(x, y)$, причем $x-1 \neq m+n$, то после выполнения 2-го блока состояние машины будет $R_n^m(x-1, y)$ и следующим должен будет выполняться 3-й блок.

3°. Если перед началом работы 3-го блока состояние машины Поста было $R_n^m(x, y)$, то выполнение этого блока закончится, после чего состояние машины Поста будет $S_n^m(x, y)$.

4°. Если перед началом работы 4-го блока состояние машины Поста было $S_n^m(x, y)$, причем $y+1 \neq m$, то после выполнения этого блока состояние машины будет $S_n^m(x, y+1)$ и следующим должен будет выполняться пятый блок.

5°. Если перед началом работы 5-го блока состояние машины Поста было $S_n^m(x, y)$, то выполнение этого блока закончится, после чего состояние машины будет $R_n^m(x, y)$.

Будем изображать символически выполнение блока прямоугольником с помещенным внутрь его номером блока; состояния, предшествующие выполнению блока и возникающие в результате этого выполнения, будем писать непосредственно левее и непосредственно правее соответствующего прямоугольника. Тогда, на основании

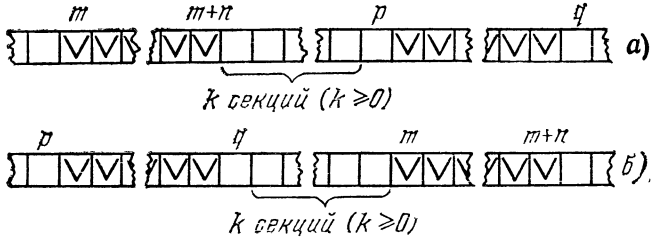


Рис. 25. Состояние из класса $W_n^m(p, q)$ имеет один из видов: *а*) (при $m + n < p$) или *б*) (при $m > q$); каретка может стоять где угодно.

утверждений 1° — 5°, работу машины Поста, исходящей из состояния H_n^m и выполняющей нашу программу, начиная со 2-го блока, можно изобразить следующей последовательностью:

$$H_n^m = R_n^m(0, 1) \quad \boxed{2}\ R_n^m(-1, 1) \quad \boxed{3}\ S_n^m(-1, 1)$$
$$\boxed{4}\ S_n^m(-1, 2) \quad \boxed{5}\ R_n^m(-1, 2) \quad \boxed{2}\ R_n^m(-2, 2)$$
$$\boxed{3}\ R_n^m(-2, 2) \quad \boxed{4}\ S_n^m(-2, 3) \quad \boxed{5}\ R_n^m(-2, 3)$$
$$\boxed{2}\ R_n^m(-3, 3) \ldots.$$

Эта последовательность продолжается до тех пор, пока не наступит одно из двух следующих событий.

Первое событие. В некоторый момент времени перед началом работы 2-го блока (быть может, в частности, в самом начале) состоянием машины оказывается такое $R_n^m(x, y)$, что $x - 1 = m + n$.

Второе событие. В некоторый момент времени перед началом работы 4-го блока состоянием машины оказывается такое $S_n^m(x, y)$, что $y + 1 = m$.

Заметим, что одно из этих событий непременно наступит (вопрос читателю: почему?).

Рассмотрим отдельно каждое из этих событий.

Первое событие. Пусть к началу работы 2-го блока состояние машины есть $R_n^m(x, y)$, где $x - 1 = m + n$. Это состояние изображено на рис. 26. После выполнения 2-го блока возникает состояние, изображенное на рис. 27. Последняя команда 2-го блока —

команда № 7 — «передает управление» 6-му блоку, т. е. именно 6-й блок должен будет теперь выполняться. Нетрудно проверить, что выполнение 6-го блока закончится и после этого возникнет состояние, показанное на рис. 28. К этому состоянию будет применяться 8-й

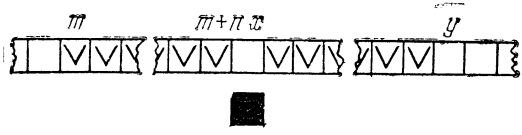


Рис. 26. Здесь показано состояние R_n^m (при $m+n+1$, y)

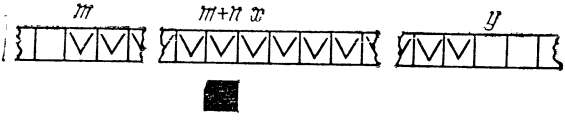


Рис. 27. Вот какое состояние возникает, если 2-й блок применить к состоянию, изображенному на рис. 26

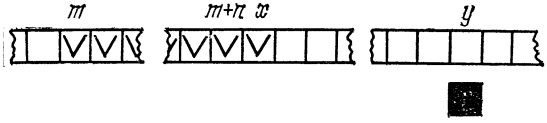


Рис. 28. Вот какое состояние возникает после выполнения 6-го блока, примененного к состоянию, изображенному на рис. 27. В этом состоянии далее произойдет результативная остановка

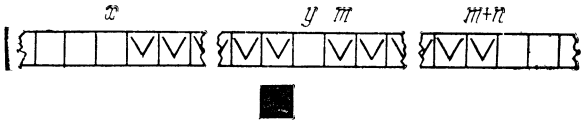


Рис. 29. Здесь изображено состояние $S_n^m(x, m-1)$

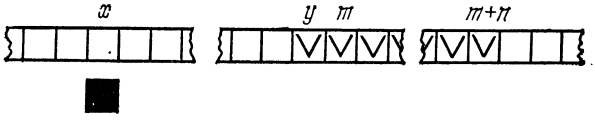


Рис. 30. В этом состоянии произойдет результативная остановка после выполнения 4-го, 7-го и 8-го блоков (в применении к состоянию, изображенному на рис. 29)

блок, который приведет к результативной остановке в этом состоянии.

Второе событие. Пусть к началу работы 4-го блока состояние машины есть $S_n^m(x, y)$, где $y+1=m$. Это состояние изображено на рис. 29. Легко проверить, что после выполнения 4-го

блока будет выполняться 7-й блок, а затем 8-й, который приведет к результативной остановке в состоянии, показанном на рис. 30.

Нам остается заметить, что как состояние из рис. 28, так и состояние из рис. 30 принадлежат классу E_{n+1}. Таким образом, в любом из двух возможных случаев работа машины закончится результативной остановкой с возникновением состояния из класса E_{n+1}, что и требовалось доказать.

§ 3. Еще о задаче прибавления единицы

Итак, общая задача о прибавлении единицы решена. Проанализируем теперь ее решение. Читатель согласится, конечно, что основная трудность, возникающая при решении этой задачи, состоит не в том,чтобы увеличить на одну метку массив длины $n+1$, записанный на ленте, а в том, чтобы найти этот массив. При решении более простых задач о прибавлении единицы, обсуждавшихся в предыдущей главе, мы знали, в каком направлении следует искать массив, и соответственно писали программу, работа которой начиналась с движения в соответствующую сторону (если только не оказывалось, что каретка в самом начале стоит на искомом массиве). В общей задаче мы не знаем, куда надо двигаться. Именно поэтому, вероятно, некоторым читателям эта задача показалась сперва неразрешимой.

Каким же способом можно «додуматься» до программы V или другой аналогичной программы, которую, возможно, читатель составит или уже составил самостоятельно? А вот каким. Представим себе, что мы стоим на бесконечной в обе стороны дороге, а где-то на этой дороге лежит волшебный камень, который мы хотим найти. Мы не знаем, где он лежит, знаем только, что где-то он непременно лежит. Как нам поступить? В какую бы сторону мы ни пошли, может случиться, что волшебный камень лежит как раз в другой стороне. Очевидно, нам надо двигаться попеременно то в одну сторону, то в другую, постоянно увеличивая размах своих колебаний: сначала делаем шаг в одну сторону, потом два шага в другую, потом три шага снова в первую сторону, потом четыре шага во вторую и т. д. до тех пор, пока не наткнемся на волшебный камень.

Задумаемся теперь вот над чем. Как определить момент, когда надо поворачивать, т. е. менять направление движения? Казалось бы, очень просто; надо только вести счет шагов и знать в каждый момент, сколько мы должны пройти шагов в данном направлении и сколько шагов из этого числа мы уже прошли. На самом деле тут

есть одна трудность, заключающаяся в том, как зафиксировать нужные нам числа шагов. Если хранить их в нашей памяти, то надо быть готовым к запоминанию сколь угодно больших чисел, что невозможно: действительно, если волшебный камень лежит достаточно далеко от нашей исходной позиции, то придется запоминать числа, превосходящие возможности нашего мозга. Можно вести запись чисел шагов, скажем, в блокнотах, носимых с собой, но тогда надо быть готовым к тому, что придется носить с собой сколь угодно объемистые записи (к тому же чтение этих записей станет самостоятельной задачей). Если запретить носить с собой какие-либо записи, то указанная трудность может показаться на первый взгляд непреодолимой. Однако есть следующий способ выхода из затруднения: можно использовать для записи саму дорогу. Мы так и поступим. Именно: следуя мальчику с пальчик, будем отмечать наш путь вдоль дороги крошками или камешками: сделав шаг, будем класть перед собой, например, камешек (если только камешек уже там не лежит). Таким образом, идя по камешкам и доходя до их конца, мы будем класть новый камешек и поворачивать в противоположную сторону — и так до тех пор, пока не найдем волшебный камень. Вот такой способ поиска и реализован в программе V; роль камешков играют здесь метки, которые ставит каретка, чтобы отметить пройденные ею секции. Заметим еще, что в нашем случае роль волшебного камня также исполняет метка, поэтому надо принять специальные меры, чтобы не спутать «камешек» с «волшебным камнем» (в программе V эти меры осуществлялись блоками проверки). «Недостаток» способа мальчика с пальчик состоит в том, что надо иметь в кармане неограниченный запас камешков или крошек (разумеется, с точки зрения машины Поста этот недостаток несущественен). Можно, впрочем, обойтись и всего двумя камешками. Для этого надо вначале положить один камешек слева от себя, а другой справа и затем ходить между ними и передвигать их; именно, каждый раз, доходя до камешка, надо переносить его на шаг вперед, повернуться и идти назад до другого камешка, с которым поступить точно так же.

Упражнение. Реализуйте посредством программы машины Поста только что изложеный способ поиска волшебного камня с помощью двух камешков. Сравните длину полученной программы с длиной программы V.

Вернемся в последний раз к рассмотрению программы V. Роль каждого блока этой программы ясна из их названия: блоки движения передвигают каретку по массиву ранее поставленных ею меток до конца этого массива, блоки проверки проверяют, не подошла ли уже каретка вплотную к искомой записи числа, блоки стирания стирают все те метки, которыми каретка отмечала свой путь. Если установить вначале состояние из класса H_n и начать осуществлять программу сразу со 2-го блока, то получится требуемый результат. (Таким образом, если потребовать, чтобы программа давала результат только в применении к состоянию из H_n, то можно было бы

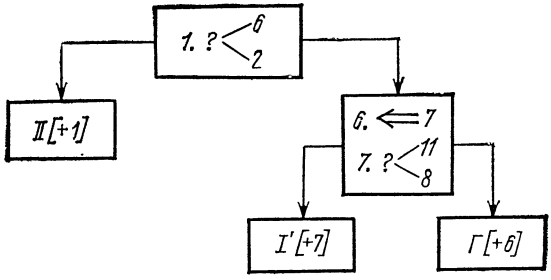


Рис. 31

ограничиться последними 19 командами, — разумеется, предварительно уменьшив все адреса и отсылки на 4.) Цель блока начала — привести машину из произвольного состояния класса E_n в какое-нибудь состояние из класса H_n. Заметим, что блок начала работает во всех случаях: даже если каретка в начальном состоянии *уже* стоит на искомой записи числа, блок начала сдвигает каретку с этого массива с тем, чтобы затем начать поиски этого же самого массива. Это кажется явно неразумным. Не проще ли сперва распознать, стоит каретка вначале на массиве или нет (т. е. не имеет ли место случай 3 на с. 45). Если стоит, то прибавить метку (так, как это требуется в задаче 2 и § 3 нашей предыдущей главы); если нет, то сдвинуться на одну секцию влево и посмотреть, не стоит ли метка там (т. е. не имеет ли место случай 2 на с. 45); если да, то прибавить к найденному массиву метку (так, как это требуется в задаче 1′ из § 2 предыдущей главы); если нет — и только в этом случае (т. е. в случае 1 на с. 45), при двух «нет», — включить команды № 5—23 из программы V.

Составим программу, реализующую этот план. Удобно составить программу сразу вместе с некоторой ее блок-схемой. Для этого обозначим через II любую из программ длины 4, служащих решением задачи 2 из предыдущей главы, через I′ — любую из программ длины 3, служащих решением задачи 1′ из той же главы, и через Г — список команд с № 5 по № 23 из программы V. Искомая программа изображена посредством своей блок-схемы на рис. 31. Напоминаем, что, согласно обозначению из § 4 предыдущей главы, для любого списка команд Ξ через Ξ[+m] обозначается список, получающийся из Ξ путем увеличения на m всех номеров команд и всех отсылок.

Программа на рис. 31 при подходящем выборе программы II (а именно в случае, если программа II содержит команду движе-

ния вправо, т. е. является одной из программ $П_1'$, $П_2'$, ..., $П_{12}'$, упомянутых в упражнении 3 из § 3 предыдущей главы), приводит к результату не более длинным путем, чем программа V, т. е. при любом начальном состоянии требует не больше, чм программа V, шагов работы машины (рекомендуем читателю проверить, что это не так, в случае когда программа II содержит команду движения влево, т. е. является одной из программ $П_1$, $П_2$, ..., $П_{12}$). В некоторых случаях программа на рис. 31 приводит к результату даже более коротким путем, чем программа V, т. е. требует меньшего числа шагов работы машины. Предлагаем читателю проверить это и убедиться, что программа на рис. 31 требует для получения результативной остановки либо столько же шагов, как программа V (для начальных состояний, подпадающих под случай 1 на с. 45), либо на 6 шагов меньше (для начальных состояний, подпадающих под случай 2), либо на 5 шагов меньше (для начальных состояний, подпадающих под случай 3).

Зато программа на рис. 31 имеет 29 команд. Даже если слить в одну три имеющиеся в этой программе команды остановки, все равно останется 27 команд, т. е. больше, чем 23. Это и не удивительно, так как здесь, по существу, имеются три различные программы для каждого из трех возможных случаев на с. 45, которые могут проявиться в начале. Роль «блока начала» — на первый взгляд несколько парадоксального — в программе V состоит как раз в сведении всех возможных случаев к одному — к состоянию из класса H_n.

Вследствии этого и получается экономия в длине программы. Здесь мы используем, таким образом, один из стандартных приемов математики — прием, состоящий в попытке свести решение любых возникающих задач к решению задач, уже решенных. (В рассмотренном примере мы сводим решение задачи о прибавлении единицы для произвольного начального состояния к решению этой же задачи для состояний из H_n.) Указанный стандартный прием приводит обычно — при успешном его проведении — к экономии различного вида, например к экономии объема памяти, необходимого, чтобы запомнить программу (а в общем случае — способ решения) того или иного класса задач. Поэтому он играет в математике очень важную роль. Недаром в математическом фольклоре бытует следующий анекдот, которым математики (по крайней мере некоторые) даже несколько гордятся; он состоит в совокупности двух задач.

Задача первая. Дано: спички, неразожженный примус, закрытый водопроводный кран, пустой чайник; требуется вскипятить воду. Решение. Зажигаем спичку, разжигаем примус, открываем водопроводный кран, наполняем чайник, ставим на примус, кипятим.

Задача вторая. Дано: разожженный примус, открытый водопроводный кран, чайник, наполненный

водой; требуется вскипятить воду. Решение. Сводим задачу к уже решенной (т. е. к первой). С этой целью тушим примус, выливаем воду из чайника, закрываем кран, после чего остается лишь выполнить действия, указанные в решении первой задачи.

Примечание. Количество действий, требующихся для сведения второй задачи к первой, можно сократить, совместив тушение примуса с выливанием воды из чайника: достаточно вылить воду на примус.

§ 4. Сложение чисел в простых случаях

Займемся теперь сложением чисел на машине Поста. В качестве слагаемых будут рассматриваться только целые неотрицательные числа. По-прежнему будем изображать целое неотрицательное число m массивом длины $m+1$. Осуществить на машине Поста сложение чисел — это значит составить такую программу, которая, будучи применена к ленте, содержащей записи чисел m_1, m_2, ..., m_k ($k \geqslant 2$), приводила бы к результативной остановке, причем после этой остановки на ленте оказывалась бы запись числа $m_1 + m_2 + \ldots m_k$. Необходимо сделать ряд уточнений. Прежде всего будем всегда предполагать, что вначале на ленте не записано ничего, кроме чисел m_1, ..., m_k, и требовать, чтобы в конце не было записано ничего, кроме их суммы; не будем налагать никаких ограничений на положение каретки в конце, что же касается ее положения вначале, то для простоты будем считать, что в начальном состоянии каретка стоит против самой левой из отмеченных секций.

Далее, можно делать различные предположения о расстоянии между массивами, служащими для записи чисел. (Расстоянием между двумя массивами естественно называть число секций, заключенных между крайней правой секцией левого массива и крайней левой секцией правого массива.) В наиболее простом случае (с которого мы и начнем) расстояние между соседними массивами равно 1. Наконец, можно накладывать те или иные ограничения на количество слагаемых, считая это количество заранее заданным или произвольным.

Наиболее простой случай рассмотрен в задаче «а».

Задача «а». Составить программу сложения двух чисел, записанных на расстоянии 1 друг от друга.

В силу сделанных предположений начальное состояние для задачи «а» будет иметь вид, как на рис. 32.

Проще всего решить эту задачу, «заполнив» отметкой пустую секцию между массивами. Тогда, если слагаемыми были числа m_1 и m_2, на ленте возникнет $m_1 + m_2 + 3$ отмеченных секций, в то время как их должно быть $m_1 + m_2 + 1$. Поэтому надо стереть две лишние

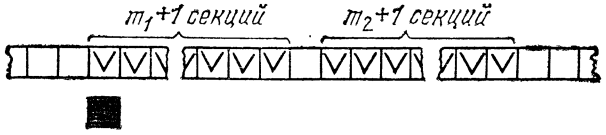


Рис. 32

метки. В выписанной ниже программе $\tilde{A}$ и реализуется сформулированный только что план действий.

Программа $\tilde{A}$

1. $\Rightarrow 2$	4. $\Rightarrow 5$	7. $\xi 8$
2. $?\begin{matrix}3\\1\end{matrix}$	5. $?\begin{matrix}6\\4\end{matrix}$	8. $\Leftarrow 9$
3. $\vee 4$	6. $\Leftarrow 7$	9. $\xi 10$
		10. стоп.

Мы получим экономию в числе команд, если сотрем две метки не в конце, а в начале, прямо из левого массива. При этом надо проявить осторожность, чтобы учесть тот случай, когда в левом массиве всего одна метка. Этот план реализуется в программе A, содержащей всего 8 команд. 3-я команда этой программы как раз и учитывает тот случай, когда в левом массиве всего одна метка.

Программа A

1. $\xi 2$	4. $\xi 5$	7. $\vee 8$
2. $\Rightarrow 3$	5. $\Rightarrow 6$	8. стоп.
3. $?\begin{matrix}8\\4\end{matrix}$	6. $?\begin{matrix}7\\5\end{matrix}$	

Задача «б». Составить программу сложения произвольного количества чисел, записанных на ленте на расстоянии 1 друг от друга.

В силу сделанных предположений начальное состояние для задачи «б» будет таким, как на рис. 33. Надо,

стало быть, составить такую программу, которая для любого числа k и любых чисел $m_1, m_2, \ldots, m_k$ давала бы результативную остановку в состоянии, в котором на ленте записано число $m_1 + m_2 + \ldots, + m_k$. Вот наиболее короткая (точнее, одна из наиболее коротких) из известных автору таких программ:

Программа Б

1.	$\xi 2$	5.	$\Rightarrow 6$	9.	$?\begin{cases}10\\8\end{cases}$
2.	$\Rightarrow 3$	6.	$?\begin{cases}7\\8\end{cases}$	10.	$\Rightarrow 11$
3.	$?\begin{cases}4\\2\end{cases}$	7.	стоп	11.	$\xi 12$
4.	$\vee 5$	8.	$\Leftarrow 9$	12.	$\Rightarrow 1.$

Рекомендуем читателю проверить эту программу (как и все остальные программы настоящего параграфа) на

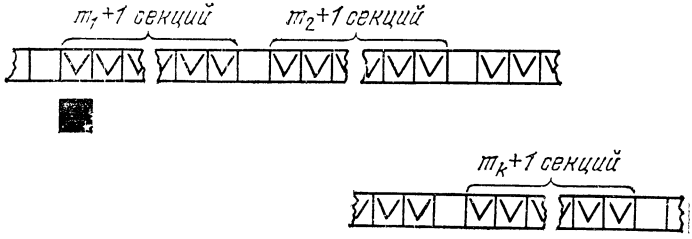


Рис. 33

каком-либо конкретном примере и тем самым понять «идею» ее работы. Эта «идея» состоит в данном случае в том, что мы последовательно отщепляем по две метки слева (командами № 1 и № 11), помещая взамен одну метку в ближайшую пустую секцию справа (командой № 4); если следующая за ближайшей справа пустой секцией также пуста (что проверяется командами № 5 и № 6), то это значит, что пора кончать (передача управления команде № 7).

Перейдем теперь к сложению чисел, записанных на ленте с произвольным расстоянием между ними.

Задача «в». Составить программу сложения двух чисел, записанных на произвольном расстоянии друг от друга.

Начальное состояние для задачи «в» показано на рис. 34. Нужно составить программу, приводящую к цели, каковы бы ни были целые неотрицательные числа m_1, m_2, l. Вот одна из таких программ:

Программа B

1.	ξ 2	6.	? 7 / 10	10.	⇐ 11
2.	⇒ 3	7.	⇐ 8	11.	? 12 / 11
3.	? 4 / 2	8.	? 9 / 7	12.	⇒ 13
4.	∨ 5	9.	⇒ 1	13.	ξ 14
5.	⇒ 6			14.	стоп.

«Идея» этой программы состоит в том, что левый массив «передвигается» направо до тех пор, пока не

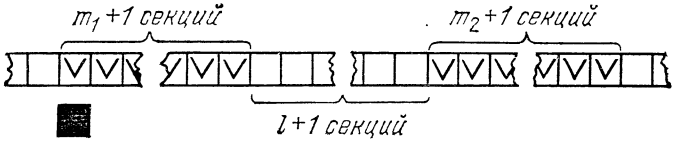


Рис. 34

сольется с правым. Передвижение массива осуществляется путем «перенесения» самой левой метки массива в ближайшую пустую секцию справа (метка отщепляется командой № 1 и ставится командой № 4). Когда массивы сливаются (что обнаруживается командами № 5 и № 6), отмеченными оказываются $m_1 + m_2 + 2$ секции, т. е. на одну больше, чем надо. Команды № 10, 11, 12 перегоняют каретку в левый конец массива, где и стирается лишняя метка (командой № 13). Таким образом (это важно для дальнейшего), в заключительном состоянии каретка стоит непосредственно левее образовавшейся суммы.

При решении задачи «а» мы уже видели, что можно сократить длину программы, если стирать лишнюю метку не в конце, а в начале работы машины. Используя этот прием, получаем следующую программу B_1 длины 12, осуществляющую тот же метод передвижения левого

массива вправо и потому очень похожую на программу B̃:

Программа B_1

1.	ξ 2	5.	⇒6	9.	?→10, 12
2.	⇒3	6.	?→7, 5	10.	⇐11
3.	?→12, 4	7.	∨ 8	11.	?→2, 10
4.	ξ 5	8.	⇒9	12.	стоп.

Более короткие, чем из 12 команд, программы, удовлетворяющие условию задачи «в», неизвестны автору. В то же время можно построить много решений задачи «в», имеющих длину 12. Прежде всего легко получить 11! программ путем образования всевозможных перестановок всех, кроме первой, команд программы B_1 (разумеется, каждая такая перестановка должна сопровождаться надлежащим изменением номеров и отсылок). Кроме того, можно пытаться искать принципиально другие методы сложения. Возможен, например, метод, состоящий в многократном осуществлении следующей процедуры: к записи левого числа приставляется справа одна метка, в компенсацию чего одна метка стирается затем в левом конце записи правого числа; эта процедура продолжается до тех пор, пока правый массив не будет исчерпан; «лишняя» метка стирается в самом начале командой № 1. Нижеследующая программа B_2 длины 12 реализует этот метод.

Программа B_2

1.	ξ 2	5.	⇒6	9.	?→12, 10
2.	⇒3	6.	?→5, 7	10.	⇐11
3.	?→4, 2	7.	ξ 8	11.	?→10, 2
4.	∨ 5	8.	⇒9	12.	стоп.

§ 5. Сложение чисел в более сложных случаях

Задача «г (k)». Для каждого k составить программу сложения k чисел, записанных с произвольными расстояниями между ними.

Начальное состояние для задачи «г (k)» показано на рис. 35. Для каждого k надо, стало быть, составить

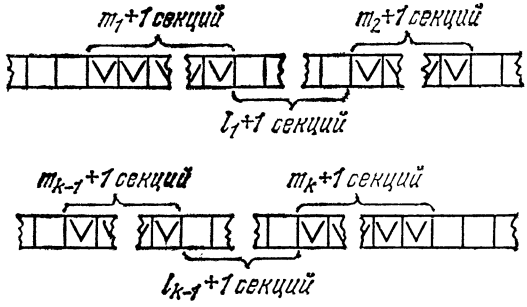


Рис. 35

программу, приводящую к цели при любых целых неотрицательных $m_1, m_2, \ldots, m_k, l_1, l_2, \ldots, l_{k-1}$.

Искомую программу мы найдем следующим образом.

Пусть Σ — какая-либо программа, являющаяся решением задачи «в» и обладающая следующими свойствами:

1) в процессе применения программы Σ к начальному состоянию из задачи «в» (т. е. такому, в котором на ленте имеются записи двух чисел и больше ничего, а каретка обозревает самую левую из отмеченных секций) каретка ни разу не заходит правее самой правой из отмеченных в начальном состоянии секций;

2) после окончания выполнения программы Σ, примененной к начальному состоянию из задачи «в», каретка оказывается стоящей против самой левой из отмеченных секций образовавшегося массива;

3) программа содержит ровно одну команду остановки, и эта команда является последней командой программы.

Пусть программа Σ содержит $a+1$ команд. Через Ξ обозначим список первых a команд из Σ. Составим такую программу:

Программа $\Gamma^*(k)$

Ξ

$\Xi[+a]$ $\Xi[+(k-2)a]$

$\Xi[+2a]$ $(k-1)a+1$. стоп.

Предоставим читателю проверить, что эта программа является решением задачи «г(k)» (действительно, если положить $s_i = m_1 + \ldots + m_i$, то легко заметить, что команды списка Ξ осуществляют сложение чисел m_1 и m_2, команды списка $\Xi[+a]$ — сложение чисел s_2 и m_3 и т. д.; команды списка $\Xi[+ia]$ — сложение чисел s_{i+1} и m_{i+2}; наконец, команды списка

$$\Xi[+(k-2)\,a]$$

осуществляют сложение чисел s_{k-1} и m_k, т. е. получение искомой суммы).

Упражнение. Каким образом при установлении того, что программа $\Gamma^*(k)$ служит решением задачи «г(k)», используются свойства 1)—3) программы Ξ?

Нам осталось найти программу Σ с требуемыми свойствами. Ни одна из программ $\tilde{B}$, B_1 и B_2 не годится. Однако программа $\tilde{B}$ обладает свойствами 1) и 3). Легко чуть-чуть изменить ее так, чтобы выполнялось и свойство 2). Достаточно вместо последней команды программы $\tilde{B}$ написать такие две:

14. $\Rightarrow$ 15 15. стоп.

В силу сделанного выше (при объяснении работы программы $\tilde{B}$) замечания о заключительном положении каретки после выполнения программы $\tilde{B}$, полученная 15-командная программа $\tilde{B}^*$ будет обладать не только свойствами 1) и 3), но и свойством 2).

В качестве программы Σ как основы для построения программы $\Gamma^*(k)$ можно взять и программу B_2^* также длины 15, получающуюся из программы B_2, если вместо последней команды этой программы написать четыре такие:

12. $\Leftarrow$ 13 14. $\Rightarrow$ 15

13. ? $\swarrow$ 14, $\searrow$ 12 15. стоп.

Программа B_2^* обладает свойством 2) и 3), а вместо свойства 1) — некоторым более слабым свойством (вопрос читателю: каким?), достаточным, однако, для того, чтобы строящаяся на основе B_2^* программа $\Gamma^*(k)$ была решением задачи «г(k)».

Если в программе Σ 15 команд, то в программе $\Gamma^*(k)$ будет $(k-1)14+1 = 14k-13$ команд.

Однако можно составить программу, отвечающую требованиям задачи «г(k)» и содержащую меньше, нежели $14k-13$, команд.

С этой целью обозначим через Δ_i следующий список команд ($i = 0, 1, 2, \ldots$):

$3+9i.$ $\Rightarrow 4+9i$

$4+9i.$? $\swarrow 3+9i$, $\searrow 5+9i$

$5+9i.$ $\xi 6+9i$

$6+9i.$ $\Rightarrow 7+9i$

$7+9i.$? $\swarrow 3+9(i+1)$, $\searrow 8+9i$

$8+9i.$ $\Leftarrow 9+9i$

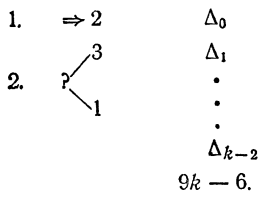
$9+9i.$? $\swarrow 8+9i$, $\searrow 10+9i$

$10+9i.$ $\Rightarrow 11+9i$

$11+9i.$ $\vee 3+9i.$

Обозначим теперь через $\Gamma(k)$ следующую программу:

Программа $\Gamma(k)$



$9k - 6$. стоп.

Предоставим читателю убедиться, что $\Gamma(k)$, во-первых, действительно есть программа и, во-вторых, программа, удовлетворяющая требованиям задачи «г(k)». При этом в программе $9k - 6$ команд. Заметим, что, положив $k = 2$, мы получим — в виде $\Gamma(2)$ — новое решение задачи «в», причем это решение будет содержать 12 команд — столько же, сколько программы B_1 и B_2. Программа $\Gamma(2)$ работает по тому же, что и программа B_2, методу, состоящему в перенесении меток из левого массива в правый; однако при выполнении программы $\Gamma(2)$, в отличие от B_2, метка сначала стирается в правом массиве, а потом присоединяется к левому, причем последняя метка правого массива стирается без того, чтобы новая метка возникла в левом.

Задача «д». Составить программу сложения произвольного количества чисел, записанных с произвольными расстояниями между ними.

Очевидно, что каждая программа, являющаяся решением задачи «д», будет одновременно служить решением каждой из предшествующих задач. Однако ни одно из приведенных выше решений задач «а», «б», «в», «г (k)» не является решением задачи «д» (проверьте это).

Начальное состояние для задачи «д» изображено на рис. 35. Однако теперь нужно составить единую программу, приводящую к цели для произвольного k. Попытайтесь найти такую программу самостоятельно. Вы увидите, что это не так просто. А может быть, требуемой программы не существует вовсе? Тогда попытайтесь доказать, что ее не существует. Ответ на вопрос, имеет ли задача «д» решение, будет дан в следующей главе.

ГЛАВА ЧЕТВЕРТАЯ

ВОЗМОЖНОСТИ МАШИНЫ ПОСТА

В настоящей главе нас будет занимать вопрос, какие вообще вычисления можно производить на машине Поста. В этой связи нам придется коснуться общего понятия алгоритма. В заключительном параграфе будет сделана попытка сопоставить машину Поста с электронными вычислительными машинами. Изложение начинается с разбора одной задачи, оставшейся нерешенной в предыдущей главе.

§ 1. О задаче сложения чисел на произвольных расстояниях

В конце предыдущей главы была поставлена наиболее общая задача сложения чисел, записанных на ленте машины Поста, — задача «д». В этой задаче требовалось найти такую единую программу машины Поста, которая осуществляла бы сложение произвольного количества чисел, записанных на ленте на произвольных расстояниях. Предполагалось, что, кроме этих чисел, на ленте ничего не записано и что каретка стоит вначале на самой левой из отмеченных секций. В конце работы программы на ленте должна быть записана сумма всех исходных чисел и больше ничего (т. е. в остальном лента должна быть пустой).

Напомним, что с лентой у нас сопоставлена «постоянная система координат», согласно которой секции ленты занумерованы целыми числами от $-\infty$ до $+\infty$. Для определенности будем считать, что все исходные числа записаны в правой, «неотрицательной» части ленты, причем самая левая из отмеченных секций имеет номер нуль. Таким образом, в начальном состоянии каретка обозревает нулевую секцию.

Не предрешая пока вопрос, как выглядит решение поставленной задачи, попробуем проанализировать такое решение.

Итак, предположим, что некоторая программа является решением задачи «д», и обозначим ее через Д. Рассмотрим состояние машины Поста, показанное на рис. 36. В этом состоянии на ленте на расстоянии единицы друг от друга записаны два числа, каждое из

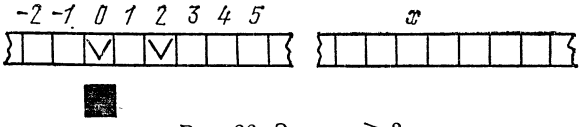


Рис. 36. Здесь $x \geqslant 3$

которых равно 0. Возьмем это состояние в качестве начального. Тогда программа Д должна привести к результативной остановке, причем на ленте будет записано число 0. За время работы машины вплоть до результативной остановки каретка сумеет побывать лишь в конечном числе секций ленты. Пусть z будет наибольшим из номеров секций, в которых побывала каретка. Обозначим через x наибольшее из чисел z и 3. Рассмотрим

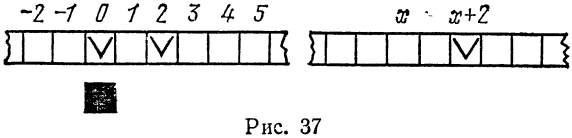


Рис. 37

теперь состояние машины Поста, в котором отмеченными являются секции 0, 2, $x+2$, а остальные пусты; каретку установим против секции № 0 (рис. 37). Возьмем это состояние в качестве начального и применим к нему программу Д. Посмотрим, как она будет работать. Легко видеть, что она будет работать «так же», как и в применении к начальному состоянию на рис. 36.

Взятое в кавычки «так же» уточняется следующим образом. Рассмотрим два экземпляра машины Поста. Будем говорить, что некоторое состояние первой машины эквивалентно некоторому состоянию второй машины, если в этих состояниях: 1) каретка первой машины стоит против секции с тем же номером, что и каретка второй машины; 2) каждая секция первой машины, расположенная левее секции № $(x+1)$, является

отмеченной или пустой, одновременно с имеющей тот же номер секцией второй машины; 3) все секции первой машины, расположенные правее секции № x, пусты; 4) у второй машины секция № $(x+1)$ пуста, секция № $(x+2)$ отмечена, а все секции, расположенные правее секции № $(x+2)$, пусты. Будем считать, что обе машины работают по программе Д синхронно, причем начальным состоянием для первой машины служит состояние, показанное на рис. 36, а для второй машины — состояние, показанное на рис. 37. Эти состояния эквивалентны. Индукцией по t легко доказывается, что в любой момент времени t будет справедливо следующее утверждение: состояния обеих машин в этот момент эквивалентны, а команда, которую должна выполнять первая машина, совпадает с командой, которую должна выполнять вторая машина. Следовательно, работая синхронно, обе машины будут переходить от эквивалентных состояний к эквивалентным под действием одних и тех же команд. Это и имелось в виду, когда говорилось, что программа в применении к состоянию на рис. 37 будет работать так же, как и в применении к состоянию на рис. 36. В какой-то момент машины одновременно придут в заключительное состояние, в котором сработает команда остановки. Обе машины одновременно результативно остановятся. На ленте первой машины будет записано число 0, т. е. будет иметься только одна отмеченная секция, причем эта секция будет расположена левее секции № $(x+1)$. Следовательно, в силу эквивалентности заключительных состояний на ленте второй машины левее секции № $(x+1)$ будет иметься одна отмеченная секция и, кроме того, отмеченной будет и сама секция № $(x+1)$, а всего будут две отмеченные секции.

Мы пришли к выводу, что при применении программы Д к состоянию на рис. 37 наступит результативная остановка в состоянии, в котором будут отмечены ровно две секции.

С другой стороны, в состоянии на рис. 37 на ленте содержится запись трех чисел, каждое из которых равно нулю (расстояния между ними равны соответственно 1 и $x-1$), в остальном лента пуста и каретка стоит против самой левой из отмеченных секций. Программа Д является, по предположению, решением задачи «д». Поэтому она должна в применении к этому начальному состоянию привести к результативной остановке в состоянии, в котором на ленте окажется записанной сумма

записанных первоначально чисел, а в остальном лента должна быть пуста. В данном случае сумма равна 0. Таким образом, в заключительном состоянии на ленте будет отмечена только одна секция. Но это находится в противоречии с полученным выше выводом.

Обнаружившееся противоречие убеждает нас в том, что программы, служащей решением задачи «д», не может существовать.

Причина, по которой задача «д» не имеет решения, очень наглядна: сколько бы каретка ни путешествовала по ленте, она, так сказать, «никогда не будет знать», обошла она уже записи всех слагаемых или, может быть, далеко вправо стоит еще одно слагаемое, до которого ей надо дойти. Поэтому нельзя составить программу, которая приводила бы к результативной остановке с гарантией, что все слагаемые учтены. Такую программу можно составить лишь для частных случаев, например, когда заранее известно число слагаемых (как в задаче «г (k)» из § 5 предыдущей главы) или когда слагаемые записаны на заданных (как в задаче «б») или хотя бы на ограниченных расстояниях друг от друга.

Упражнение. Для каждого q составьте программу сложения произвольного количества чисел, записанных на произвольных, но не превосходящих q, расстояниях друг от друга.

На примере задачи «д» мы видим, сколь важен способ записи исходных данных на ленте.

§ 2. Предложение (постулат) Поста

В предыдущих главах читатель приобрел некоторый опыт программирования на машине Поста. Чтобы закрепить этот опыт, рекомендуем выполнить упражнения 1—5. В них читателю предлагается выбрать самому начальное положение каретки, а в упражнениях 2, 3, 5 еще и расстояние, на котором записываются исходные числа. Числа всюду в данной главе подразумеваются целыми неотрицательными.

Упражнение 1. Составить программу деления чисел на 2, на 3, на 4, ..., на заданное число k. Пояснение. Под делением понимается нахождение частного или неполного частного, так что результат деления 7 на 3 будет 2.

Упражнение 2. Составить программу вычитания одного числа из другого. Пояснение. Если вычитание

невозможно (вычитаемое больше уменьшаемого), программа не должна приводить к результативной остановке.

Упражнение 3. Составить программу умножения двух чисел.

Упражнение 4. Составить программу возведения чисел в квадрат.

Упражнение 5. Составить программу деления одного числа на другое. Пояснение. Если делитель есть 0, то программа не должна приводить к результативной остановке.

Для дальнейшего нам понадобится одно фундаментальное, хотя и простое, понятие — понятие числового кортежа. Часто приходится иметь дело не с отдельными числами, а с упорядоченными парами чисел; при этом допускается, что оба члена пары совпадают. Записывать упорядоченную пару мы будем так: сперва первый член пары, затем второй, и все это обрамлять угловыми скобками. Так, например, ⟨33, 4⟩ и ⟨2, 2⟩ суть упорядоченные пары. Аналогично вводится понятие упорядоченной тройки. Возможны такие, например, упорядоченные тройки чисел: ⟨2, 5, 1⟩; ⟨6, 6, 6⟩; ⟨4, 5, 4⟩. А ⟨3, 8, 7, 8, 8, 2, 5⟩ — упорядоченная семерка чисел. Упорядоченный набор чисел любой длины называется *числовым кортежем*; так, выписанные только что упорядоченные пары, упорядоченные тройки и упорядоченная семерка суть кортежи. Число «мест» кортежа называется его *длиной*. Так, кортеж ⟨6, 2, 2, 6⟩ имеет длину 4. Кортежи длины 2 — это упорядоченные пары, кортежи длины 3 — это упорядоченные тройки и т. д.

Все задачи и большинство упражнений на составление программ, которыми мы занимались в данной книге имели следующий вид. Фиксировался некоторый класс исходных данных: числа (как в упражнениях 1 и 4 из настоящего параграфа или в задаче о прибавлении единицы из главы 2), числовые кортежи заданной длины (длины два, как в упражнениях 2, 3 и 5 из настоящего параграфа и в задачах «а», «в» и «г(2)» из главы 3, или вообще длины k, как в задаче «г(k)» из названной главы), числовые кортежи произвольной длины (как в задачах «б» и «д» из главы 3).

Каждому исходному данному из выбранного класса — числу или кортежу — либо ставилось в соответствие некоторе результирующее число (квадрат исходного числа, или частное от деления первого члена исход-

ного кортежа на второй, или сумма всех членов исходного кортежа и т. п.), либо ничего не ставилось в соответствие (например, частное или неполное частное ставилось в соответствие только той паре, у которой второй член отличен от нуля). Требовалось составить программу, которая бы запись любого исходного данного на ленте преобразовывала — с результативной остановкой — в запись результирующего числа, если такое существует; в применении же к записи исходного данного, для которого не определено результирующее число, программа не должна давать результативной остановки. Для простоты всегда будем подразумевать, что в начальном состоянии на ленте записано только исходное данное и каретка стоит против самой левой из отмеченных секций и что в заключительном состоянии на ленте также записано результирующее число, а каретка стоит где угодно. Необходимо еще уточнить, как записывать числовые кортежи. В предыдущем параграфе мы видели, что от способа записи кортежа, а именно от расстояний между его членами, зависит само существование требуемой программы. Условимся записывать кортеж, помещая записи его членов на расстоянии единица друг от друга. При так фиксированных начальном положении каретки и способе записи чисел и кортежей можно говорить сокращенно о применении программы к числу или кортежу и переработке его в число или кортеж, имея при этом в виду на самом деле записи чисел и кортежей.

Поставим теперь следующий естественный вопрос. В каких случаях задачи описанного только что вида на составление программы имеют решения? Иными словами, какие вычисления можно осуществлять на машине Поста?

Ответ на этот вопрос таков. Задача на составление программы, приводящей от исходного данного к результирующему числу (и не приводящей ни к какому результату, если результирующего числа не существует), тогда и только тогда имеет решение, когда имеется какой-нибудь общий способ, позволяющий по произвольному исходному данному выписать результирующее число и не выдающий никакого — заведомо ложного в этом случае — результата, коль скоро результирующего числа не существует. Это дающее ответ на наш вопрос утверждение будем называть — по имени его автора — *предложением*, или *постулатом*, *Поста*. Подчеркнем, что в постулате Поста речь идет о некотором едином способе (так

же как о единой программе), общем для всех исходных данных.

Пример 1. Задача: «Составить программу машины Поста для возведения в куб» разрешима, потому что существует общий способ, позволяющий вычислять n^3 по n.

Пример 2. По такой же причине существует программа машины Поста, переводящая запись кортежа $\langle x, y, z\rangle$ в запись числа $(x^y + zy)\,(z^2 - y)$. Эта программа (как и соответствующий способ вычислений) не приводит ни к какому результату при $z^2 < y$ (все числа, с которыми мы имеем дело, — целые неотрицательные!).

Пример 3. Пусть требуется составить программу машины Поста, обладающую следующим свойством: если в десятичном разложении числа π встречается $n + 2$ идущих подряд десятичных знаков, среди которых лишь первый и последний отличны от девятки, а остальные n равны девятке, то программа перерабатывает это число n в 1; если же описанной конечной последовательности из $n + 2$ знаков в разложении π не встречается, программа перерабатывает n в 0. В данном случае, хотя каждому n и поставлено в соответствие некоторое результирующее число ξ_n, равное 0 или 1, нам не известен общий способ, позволяющий вычислять это ξ_n для произвольного n. Анализ первых 800 знаков разложения π [1]) показывает лишь, что $\xi_n = 1$ для $n = 0, 1, 2, 6$. Вообще, сколько бы ни выписать десятичных знаков разложения π, из полученной последовательности знаков мы можем извлечь лишь информацию о равенстве ξ_n единице при некоторых значениях n и никаких сведений о равенстве ξ_n нулю хотя бы при каком-то n. Равенство $\xi_n = 0$ при каком бы то ни было n, если и может быть установлено, то лишь косвенным способом. Общий способ вычисления ξ_n при любом n нам неизвестен. Если такого способа не существует, то в силу предложения Поста не может существовать и требуемой программы. (Заметим, что применение предложения Поста в эту сторону очевидно: если бы существовала требуемая программа, то она сама и давала бы некоторый общий способ вычисления результирующего числа по любому n.) Если же такой способ существует, то в силу предложения Поста (применение которого в эту сторону уже не очевидно) существует и требуемая программа.

---

[1]) Эти знаки приведены на с. 63 книги: Литцман В. Великаны и карлики в мире чисел. — М.: Физматгиз, 1959.

Пример 4. Определим для каждого n число ζ_n следующим способом: $\zeta_n = 1$, если при любом k в десятичном разложении π встречаются k идущих подряд девяток (подобно тому как встречаются 0, 1, 2, 3, 4, 5, 6 таких девяток); $\zeta_n = 0$ в противном случае, т. е. если существует наибольшая длина массива идущих подряд девяток (эта наибольшая длина, если и существует, то наверняка больше 5). Спрашивается, существует ли программа машины Поста, перерабатывающая каждое n в ζ_n. Ответ на этот вопрос может показаться несколько неожиданным: «Да, существует, хотя мы и не можем ее указать». В самом деле, либо в разложении π можно встретить любое число идущих подряд девяток, либо это не так. В каждом из этих случаев существует общий способ вычисления числа ζ_n: надо положить $\zeta_n = 1$ (для всех n) в первом случае и $\zeta_n = 0$ (для всех n) во втором случае. Другое дело, что мы не знаем, какой из этих двух общих способов выбрать, поскольку мы не знаем, который из двух случаев относительно разложения π имеет место в действительности. Для каждого из этих общих способов вычисления числа ζ_n существует соответствующая программа, осуществляющая вычисление ζ_n на машине Поста — для первого общего способа ($\zeta_n \equiv 1$) — программа

1.	$\xi\, 2$	4.	$\Leftarrow 5$
2.	$\Rightarrow 3$	5.	$\vee\, 6$
3.	$?\begin{cases} 4 \\ 1 \end{cases}$	6.	$\Leftarrow 7$
		7.	$\vee\, 8$
		8.	стоп,

а для второго общего способа ($\zeta_n \equiv 0$) — программа

1.	$\Rightarrow 2$	3.	$\xi\, 1$
2.	$?\begin{cases} 4 \\ 3 \end{cases}$	4.	стоп.

Мы можем с уверенностью утверждать, что одна из выписанных двух программ отвечает предъявляемым требованиям, но современное состояние наших знаний не позволяет нам сказать, какая именно.

Существование соответствующего единого способа вычисления было известным в примерах 1, 2 и 4 и

оставалось неизвестным в примере 3. Построены примеры, в которых такого способа заведомо не может существовать, но они слишком сложны, чтобы их можно было здесь привести.

§ 3. Машина Поста и алгоритмы

Понятие единого способа вычисления, общего для целого класса возможных исходных данных, является одним из важнейших понятий математики — как «теоретической», так и «бытовой». Именно на нем основано, например, обучение математике в начальной школе. Многочисленные примеры на четыре действия с многозначными числами не преследуют ведь цель научить складывать, вычитать, умножать и делить именно те числа, которые встречаются в этих примерах. Цель этих упражнений — отработать способы сложения, вычитания, умножения и деления столбиком, имея, конечно, в виду, что эти способы применимы к *любым* упорядоченным парам чисел, а не только к тем, которые встречаются в задачнике. Когда говорят, что ученик умеет складывать, разумеют под этим не то, что он для любой пары чисел найдет рано или поздно их сумму, а то, что он владеет общим способом сложения.

Столь важное понятие нуждается, конечно, в специальном термине для своего наименования: в качестве такого термина употребляется слово «алгоритм» (другое написание: «алгорифм»). Мы можем сказать теперь, что в примерах 1, 2 и 4 существуют алгоритмы (хотя мы и не знаем, каков он для примера 4), в примере 3 алгоритм неизвестен (и неизвестно даже, существует ли он). Как уже отмечалось, известны случаи, когда требуемого алгоритма заведомо не существует.

Хотя понятие алгоритма часто встречается в практике и в науке, оно не всегда замечается: само слово «алгоритм» все еще недостаточно популярно; общеизвестным является лишь словосочетание «алгоритм Евклида», служащее для обозначения одного из алгоритмов для нахождения наибольшего общего делителя. Мы нередко оказываемся тем самым в положении мольерова мещанина во дворянстве, уже в зрелом возрасте впервые узнавшего, что он всю жизнь говорил прозой. Как бы то ни было, теперь мы знаем, что в школе нас учили именно алгоритмам. (Популярная экскурсия в область теории алгоритмов предпринята в статье

В. А. Успенского, А. Л. Семенова «Решимые и нерешимые алгоритмические проблемы (см. с. 90—96 настоящей книги).)

Предполагается, что для каждого алгоритма указана некоторая совокупность, называемая *областью* его *возможных исходных данных* и состоящая из всех объектов (например, чисел или кортежей), к которым рассматриваемый алгоритм имеет смысл пытаться применять. В применении к какому-либо из своих возможных исходных данных алгоритм может как давать результат (в этом случае говорят, что алгоритм *применим* к этому исходному данному и перерабатывает его в результат), так и не давать результата (в этом случае говорят, что алгоритм *неприменим* к этому исходному данному). Так, для алгоритма вычитания столбиком исходными данными служат упорядоченные пары чисел, а алгоритм применим только к тем из них, у которых второй член не больше первого. Вряд ли целесообразно считать, что для этого алгоритма класс возможных исходных данных состоит лишь из пар $\langle a, b\rangle$, у которых $a \geqslant b$. Ведь возможно пытаться произвести вычитание второго числа из первого и для пары

$$\langle 1244444445444445,\ 1244444454444445\rangle;$$

просто в этом случае мы не получим никакого результата.

Посмотрим, как понятие алгоритма связано с машиной Поста. Рассмотрим такие программы машины Поста, которые в применении к любому числу либо вовсе не дают результативной остановки, либо дают в качестве результата снова число (т. е. если происходит результативная остановка, то на ленте оказывается записанным некоторое число и только оно). Каждая такая программа задает следующий алгоритм, областью возможных исходных данных которого служит совокупность всех целых неотрицательных чисел и все результаты суть также числа: надо взять исходное число, записать его на ленте машины Поста, поставить каретку против самой левой из отмеченных секций, пустить машину в ход согласно рассматриваемой программе, дождаться результативной остановки и прочесть число, которое после этой остановки окажется записанным на ленте.

Фиксируем теперь какое-либо число k и рассмотрим такие программы, которые в применении к любому числовому кортежу длины k либо вовсе не дают результа-

тивной остановки, либо дают в качестве результата число. Каждая такая программа задает следующий алгоритм, областью возможных исходных данных которого служит множество всех числовых кортежей длины k, а все результаты суть числа: надо взять исходный числовой кортеж, записать его на ленте машины Поста, поставить каретку против самой левой из отмеченных секций, пустить машину в ход согласно рассматриваемой программе, дождаться результативной остановки и прочесть число, которое после этой остановки окажется записанным на ленте.

Аналогично, каждая программа машины Поста, которая в применении к любому числовому кортежу *любой длины* либо не дает результативной остановки, либо дает в качестве результата число, задает некоторый алгоритм.

Обозначим натуральный ряд (т. е. совокупность всех целых неотрицательных чисел) буквой N, множество всех числовых кортежей — символом N^∞, множество всех числовых кортежей длины k — символом N^k.

Теперь с помощью термина «алгоритм» предложение Поста можно сформулировать так.

Пусть дан некоторый класс исходных данных — N, N^k или N^∞. Пусть каждому исходному данному из этого класса либо ничего не поставлено в соответствие, либо поставлено в соответствие некоторое (вообще говоря, свое для каждого исходного данного) результирующее число. Рассмотрим две задачи: (П) «Составить программу машины Поста, перерабатывающую любое исходное данное, для которого есть результирующее число, в это число и не приводящую ни к какому результату для исходного данного, для которого нет результирующего числа»; (А), получающуюся из (П) заменой слов «программа машины Поста» на слово «алгоритм». Тогда задачи (П) и (А) одновременно либо не имеют, либо имеют решения.

До сих пор мы никак не обосновывали предложение Поста, а принимали его на веру. Мы увидим сейчас, что в какой-то степени это неизбежно. Действительно, предложение Поста состоит, по существу, из двух утверждений: в первом говорится, что из разрешимости задачи (П) следует разрешимость задачи (А), во втором говорится, что из разрешимости задачи (А) следует разрешимость задачи (П). Первое из этих утверждений очевидно, поскольку программа, служащая решением за-

дачи (П), сама по себе образует алгоритм, служащий решением задачи (А); об этом уже говорилось при разборе примера 3. Что касается второго утверждения, которое только и нуждается в обосновании (и к которому сводится, по существу, предложение Поста), то оно было сформулировано, в несколько иных выражениях, Постом в его знаменитой статье «Финитные комбинаторные процессы — формулировка 1» в качестве «рабочей гипотезы».

Эта «рабочая гипотеза», которую будем называть также гипотезой Поста, не может быть доказана — во всяком случае при том понимании слова «доказательство», которое принято в математике, — и отнюдь не потому, что она неверна, а потому, что участвующее в ней общее понятие алгоритма не является «математически» определенным. Вспомним, что на с. 66 мы определили алгоритм как «единый способ вычисления, общий для целого класса возможных исходных данных». Известны и другие аналогичные определения: алгоритм, или алгорифм, — это «всякая система вычислений, выполняемых по строго определенным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи» [1]), это — «вычислительный процесс, совершаемый согласно точному предписанию и ведущий от могущих варьировать исходных данных к искомому результату» [2]), это — точное предписание, определяющее вычислительный процесс, ведущий от варьируемых исходных данных к искомому результату [3]).

Разумеется, все это не математические определения, а, скорее, описания понятия алгоритма. (В отношении отсутствия исчерпывающего «строгого» определения понятие алгоритма напоминает понятие доказательства; впрочем, понятие доказательства не имеет не только определения, но и удовлетворительных описаний [4]).)

---

[1]) Колмогоров А. Н. Алгоритм//Большая Советская Энциклопедия. — 2-е изд. — Т. 2. — М.: Советская энциклопедия, 1950. — С. 65.

[2]) Марков А. А. Теория алгорифмов//Тр. Мат. ин-та АН СССР им. В. А. Стеклова. — Т. 38. — М.: Изд-во АН СССР, 1951. — С. 176.

[3]) Марков А. А. Теория алгорифмов//Тр. Мат. ин-та АН СССР им. В. А. Стеклова. — Т. 42. — М.: Изд-во АН СССР, 1954. — С. 3.

[4]) И сейчас, во второй половине XX в., как и во времена Евклида, математическое доказательство продолжает оставаться не более чем убедительным рассуждением, которое в состоянии убедить нас

На современной стадии развития математики не представляется возможным «строго доказывать» гипотезу Поста, опираясь на такого рода описания алгоритма.

Обоснование «рабочей гипотезы» Поста происходит на другом пути, более привычном для естествоиспытателя, чем для математика, — на пути эксперимента, подкрепляемого умозрительными рассуждениями. Эксперимент показывает, что, действительно, всякий раз, как нам указывают алгоритм, этот алгоритм можно перевести в форму приводящей к тому же результату программы машины Поста. Умозрительные рассуждения — мы не будем их здесь приводить — подтверждают, что те представления, которые сложились сейчас у математиков о понятии способа вычисления, т. е. алгоритма, не дают возможности построить алгоритм, который нельзя было бы заменить программой машины Поста. Сам Пост видел успех своих идей в том, чтобы превратить свою «рабочую гипотезу» в «закон природы». Можно считать, что это превращение уже произошло.

В заключение этого параграфа дадим еще одну формулировку гипотезы Поста. Рассмотрим какие-либо два алгоритма, у которых совпадает совокупность их возможных исходных данных. Назовем такие два алгоритма *равносильными*, коль скоро к любому исходному данному из общей для них совокупности этих данных они либо оба применимы, либо оба неприменимы, и если применимы, то дают один и тот же результат. Понятие равносильности позволяет сформулировать гипотезу Поста следующим образом: каждый алгоритм, все результаты которого суть числа, а областью возможных исходных данных служат N, N^k или N^∞, равносилен алгоритму с той же совокупностью возможных исходных данных, задаваемому некоторой программой машины Поста (задаваемому так, как объяснено выше, на с. 67 и 68).

настолько, что мы становимся готовы убеждать с его помощью других. Формализация понятия доказательства средствами математической логики, при всей своей важности, не позволяет полностью устранить употребление этого понятия в его описанном только что интуитивном понимании. Великое открытие, сделанное в 30-х годах выдающимся математиком и логиком нашего времени Куртом Гёделем, состоит в том, что попытка формально уточнить понятие доказательства неизбежно оказывается неполной: обнаруживаются истины, доказываемые интуитивно, но не допускающие доказательства в рамках произведенного уточнения. (См.: Успенский В. А. Теорема Гёделя о неполноте. — М.: Наука, 1982.)

§ 4. Вычислимые функции и тезис Поста

Этот параграф посвящен одному из центральных понятий теории алгоритмов — понятию вычислимой функции. Он предъявляет несколько большие требования к подготовке читателя — именно: предполагается знакомство с понятиями множества и соответствия.

Прежде всего отметим, что *вычислимая функция* — это функция, которая вычислима [1]). Теперь нужно объяснить слова «функция» и «вычислима».

Функцией из множества A в множество B называется соответствие между элементами A и B, при котором каждому элементу A соответствует не более одного элемента B (но может и не соответствовать ни одного). Множество A называется при этом *областью отправления*, а множество B — *областью прибытия* рассматриваемой функции. Иногда говорят о функции как о «правиле» или «законе», позволяющем сопоставить с некоторыми элементами A соответствующие элементы B, но эти термины, пожалуй, менее удачны, так как могут создать неверное представление о том, что правило (закон) должно (или может) быть как-то записано. В действительности же это не так: какой бы способ описания функций — например, с помощью предложений на русском языке, дополненном, если нужно, математическими символами, — мы ни выбрали, всегда найдутся функции, не имеющие описания [2]).

Мы будем рассматривать в качестве области отправления одно из множеств N, N^k и N^∞ (впрочем, мы можем отождествить натуральные числа с кортежами длины 1 и — тем самым — не различать множества N и N^1), а в качестве области прибытия — множество N. Таким образом, аргументами рассматриваемых нами функций будут служить натуральные (т. е. целые неотрицательные) числа и числовые кортежи, а значениями — натуральные числа. В качестве примера функции такого рода можно привести функцию «вычитание». Эта функция из N^2 в N сопоставляет с парой $\langle x, y\rangle$ число $x - y$, если $x \geqslant y$, и не сопоставляет ничего, если $x < y$. Функции

---

[1]) Это замечание не так тривиально, как кажется: к примеру, милостивый государь — вовсе не государь, который милостив! Ср. также выражение «лошадиная сила».

[2]) Этот факт может быть доказан примерно таким же способом, каким будет ниже установлено существование невычислимой функции.

из N^k в N обычно называются *числовыми*. Теперь мы должны объяснить, какие функции мы будем считать вычислимыми. При этом объяснении нам придется пользоваться понятием алгоритма. Поэтому понятие вычислимой функции окажется не более (но и не менее) «математически» определенным, чем понятие алгоритма.

Итак, пусть имеется алгоритм $\boldsymbol{A}$, областью возможных исходных данных для которого служит либо множество N^k при некотором $k \geqslant 1$, либо множество N^∞. Пусть результат применения алгоритма $\boldsymbol{A}$ к любому возможному исходному данному (если только такой результат существует) всегда является натуральным числом. Тогда мы можем определить функцию f из N^k в N (если областью возможных данных служило N^k) или из N^∞ в N (если областью возможных исходных данных служило N^∞) так: тем кортежам $\langle x_1, \ldots, x_k\rangle$, для которых результат применения алгоритма $\boldsymbol{A}$ (к ним) существует, функция ставит в соответствие этот результат, а остальным кортежам функция не ставит в соответствие ничего. Другими словами,

$$f(x_1, \ldots, x_k) = \begin{cases} \text{результат применения } \boldsymbol{A} \text{ к } \langle x_1, \ldots, x_k\rangle, \\ \qquad \text{если } \boldsymbol{A} \text{ применим к } \langle x_1, \ldots, x_k\rangle, \\ \text{не определено, если } \boldsymbol{A} \text{ неприменим} \\ \qquad \text{к } \langle x_1, \ldots, x_k\rangle. \end{cases}$$

Про так построенную функцию f говорят: «*функция f вычисляется алгоритмом $\boldsymbol{A}$* или *алгоритм $\boldsymbol{A}$ вычисляет функцию f*». Очевидно, каждый алгоритм вычисляет только одну функцию. Разумеется, разные алгоритмы могут вычислять одну и ту же функцию.

Теперь мы можем дать

Определение вычислимой функции. Функция f из N^k в N или из N^∞ в N называется *вычислимой*, если существует алгоритм, ее вычисляющий.

Согласно этому определению рассмотренная выше функция «вычитание» является вычислимой; соответствующий алгоритм «вычитание столбиком» изучается в начальной школе.

Сформулированное определение понятия вычислимой функции является скорее естественно-научным, нежели математическим. Если мы считаем это недостатком и желаем этот недостаток устранить, то должны заменить общее (и несколько неопределенное) понятие алгоритма на точное (но зато более узкое) понятие «алгоритма спе-

циального вида». В качестве таких «алгоритмов специального вида» можно взять программы для машин Поста. Тогда мы получим такое

Определение вычислимой на машине Поста функции. Функция f из N^k в N или из N^∞ в N называется *вычислимой на машине Поста* (или *вычислимой по Посту*), если существует программа p машины Поста, вычисляющая функцию f, т. е. обладающая такими свойствами:

(1) если $f(x_1, \ldots, x_k) = y$, то программа p, применяемая к исходному данному $\langle x_1, \ldots, x_k \rangle$, кончает работу, после чего на ленте остается запись числа y;

(2) если $f(x_1, \ldots, x_k)$ не определено, то применение программы p к исходному данному не приводит к результативной остановке.

Теперь предложение Поста можно сформулировать так:

Пусть дан некоторый класс исходных данных: N, N^k ($k > 1$) или N^∞; обозначим этот класс через E. Пусть задана функция f из E в N. Рассмотрим две задачи: (П) «Составить программу машины Поста, вычисляющую функцию f»; (А) «Составить алгоритм, вычисляющий функцию f». Тогда задачи (П) и (А) одновременно либо имеют, либо не имеют решения.

Используя понятие вычислимой функции и понятие вычислимой на машине Поста функции, можно это же утверждение переформулировать так:

Тезис Поста. Пусть E — одно из множеств N, N^k и N^∞. Тогда класс вычислимых функций из E в N совпадает с классом вычислимых на машине Поста функций из E в N.

Объясним, почему эту формулировку предложения Поста мы назвали «тезисом». Это объясняется следующими историческими причинами.

Мы сталкиваемся здесь со следующей ситуацией. Имеется точное математическое определение некоторого класса числовых функций (в данном случае класса вычислимых на машине Поста функций). После этого утверждается, что всякая вычислимая (в смысле «естественно-научного» определения) функция принадлежит этому классу. Впервые утверждение такого рода было сделано Алонзо Чёрчем (Alonzo Church) в 1936 г. и получило впоследствии название «тезиса Чёрча». Этот тезис утверждал, что всякая всюду определенная вычислимая числовая функция является общерекурсивной,

причем класс общерекурсивных функций был описан посредством строгого математического определения. В дальнейшем Стивн Клини (Stephen Kleene) обобщил это утверждение, выдвинув такой тезис («тезис Чёрча — Клини»): всякая (не обязательно всюду определенная) вычислимая функция частично рекурсивна[1]). Тезис Поста отличается от тезиса Чёрча — Клини тем, что вместо частично рекурсивных функций в нем рассматриваются функции, вычислимые на машинах Поста. Хотя оба эти утверждения носят «естественно-научный» характер, их эквивалентность оказывается возможным доказать математически: именно можно доказать, что всякая частично рекурсивная функция вычислима на машине Поста и, напротив, всякая вычислимая на машине Поста функция частично рекурсивна.

Таким образом, тезис Поста оказался эквивалентным тезису Чёрча — Клини. В этом факте можно усмотреть дополнительный аргумент в пользу их справедливости. Известны и другие утверждения, аналогичные тезису Поста. Они возникают, если в качестве «алгоритмов специального вида» брать не программы машин Поста, а алгоритмы из другого четко очерченного класса («машины Тьюринга», «нормальные алгорифмы Маркова», «алгоритмы Колмогорова» и др.) Все эти утверждения оказываются эквивалентными друг другу.

Теперь займемся таким вопросом: все ли числовые функции вычислимы? Если все, то теория вычислимых функций лишается своего предмета. К счастью (по крайней мере для этой теории), это не так — как уже отмечалось, существуют функции, не являющиеся вычислимыми, и мы вскоре это увидим.

---

[1]) Частично рекурсивными называются числовые функции, которые могут быть получены из тождественно нулевой функции и функции прибавления единицы ($f(x) = x + 1$) с помощью операций: 1) подстановки (функций в функцию); 2) рекуррентного (индуктивного, рекурсивного) определения функции и 3) неявного задания функции. Частично рекурсивные функции, вообще говоря, не всюду определены, поскольку неявное задание [т. е. переход от функции $f(x_1, \ldots, x_k, y)$ к функции $\varphi(x_1, \ldots, x_k)$ такой, что $f(x, \ldots, x_k, \varphi(x_1, \ldots, x_k)) = 0$] может приводить к функции (φ), не являющейся всюду определенной. Общерекурсивные функции могут быть охарактеризованы как такие частично рекурсивные функции, которые всюду определены. Подробнее см. в статье «Рекурсивные функции» в Большой Советской Энциклопедии (2-е издание, т. 36; 3-е издание, т. 21).

Но сначала — чтобы познакомиться с понятием вычислимой функции — еще раз рассмотрим примеры из § 2 настоящей главы.

В примере 1 объясняется, что функция f из N в N, для которой $f(n) = n^3$, вычислима.

В примере 2 рассматривается функция g из N^3 в N, для которой

$$g(x, y, z) = \begin{cases} (x^y + zy)(z^2 - y), & \text{если } z^2 \geqslant y, \\ \text{не определено, если } z^2 < y, \end{cases}$$

и отмечается ее вычислимость.

В примере 3 рассматривается функция

$$h(n) = \begin{cases} 1, & \text{если в десятичном разложении } \pi \text{ есть отрезок из } n \text{ девяток, окруженный недевятками,} \\ 0 & \text{в противном случае} \end{cases}$$

и говорится, что неизвестно, вычислима эта функция или нет.

В примере 4 рассматривается функция

$$q(n) = \begin{cases} 1, & \text{если при любом } k \text{ в разложении } \pi \text{ встречается отрезок из } k \text{ девяток,} \\ 0, & \text{если это не так,} \end{cases}$$

и доказывается, что эта функция вычислима.

Замечание. Если приведенное при разборе примера 4 доказательство не уничтожило полностью ваших сомнений в вычислимости этой функции q, подумайте, в каком именно из четырех следующих утверждений вы сомневаетесь.

А. Функция v, определенная так: $v(n) = 0$ при всех n, вычислима.

Б. Функция w, определенная так: $w(n) = 1$ при всех n, вычислима.

В. Функция q либо равна v, либо равна w ($q = w$, если в разложении π есть сколь угодно длинные отрезки из девяток, и $q = v$ в противном случае).

Г. Из А, Б, В следует, что функция q вычислима. Если вы сомневаетесь в утверждении В, то вы единомышленник замечательного голландского математика Л. Э. Я. Брауэра (L. E. J. Brouwer) и основанной им школы интуиционистов. (Об интуиционизме см., например: Гейтинг А. Интуиционизм, — М.: Мир, 1965; Драгалин А. Г. Математический интуиционизм.

Введение в теорию доказательств. — М.: Наука, 1979. Часть 1, п. 1 «Неформальные пояснения».)

Итак, функции из примеров 1, 2, 4 § 2 этой главы вычислимы. Если мы верим в истинность гипотезы Поста, то должны верить и в то, что эти функции вычислимы на машине Поста. И это действительно оказывается так — можно написать программы для машины Поста, вычисляющие эти функции. Например, для функции из примера 4 искомой программой является одна из двух программ, приведенных в конце § 2 (хотя неизвестно какая именно).

Теперь, привыкнув к понятию вычислимой функции, мы хотим построить пример числовой функции, не являющейся вычислимой. Более точно, мы хотим построить пример числовой функции, не вычислимой на машине Поста. Для этого нам понадобится такое утверждение:

Существует последовательность p_0, p_1, ... программ машины Поста, содержащая все возможные программы машины Поста.

(Переформулировка для знатоков: множество программ машины Поста не более чем счетно.)

Доказательство этого утверждения очень просто. В самом деле, количество программ машины Поста данной фиксированной длины конечно (см. упражнение 1 из § 2 гл. 1). Выписывая по очереди сначала все программы длины 1 (в любом порядке), затем все программы длины 2 (в любом порядке) и т. д., получаем искомую последовательность.

Теперь приступим к построению функции f из N в N, не вычислимой на машине Поста. Другими словами, нам надо построить такую функцию f, чтобы никакая программа машины Поста ее не вычисляла. Сделаем так: пусть f переводит число i куда угодно, только не в результат применения программы p_i к числу i. Более точно, определим f, например, так:

$$f(n) = \begin{cases} 0, & \text{если применение программы } p_n \text{ к числу } n \text{ не приводит к результативной остановке или результат не есть запись натурального числа;} \\ k+1, & \text{если применение программы } p_n \text{ к числу } n \text{ дает результат, являющийся записью числа } k. \end{cases}$$

Докажем, что так построенная всюду определенная функция не вычислима на машине Поста. В самом деле,

пусть p — любая программа машины Поста. Программа p встречается среди программ $p_0, p_1, \ldots$, т. е. $p = p_s$ при некотором s. Что получается при применении программ p_s и функции f к числу s? Возможны два случая.

Первый случай. Применение программы p_s к записи числа s приводит к результативной остановке, и результат есть запись натурального числа k. Тогда $f(s) = k + 1$ и, так как $k \neq k + 1$, программа p_s не вычисляет f.

Второй случай. Применение программы p_s к записи числа s не приводит к результативной остановке или результат не есть запись никакого натурального числа. Тогда p_s не вычисляет f уже потому, что функция f всюду определена.

Итак, в обоих случаях выясняется, что программа $p\,(= p_s)$ не вычисляет функции f. Что и требовалось доказать.

С приведенного только что рассуждения начинается теория вычислимых функций. Эту теорию можно развивать чисто математически, не говоря ничего об естественно-научном понятии алгоритма и не ссылаясь на тезис Поста, развивать как теорию функций, вычислимых на машине Поста.

Зачем же тогда нужен тезис Поста? А вот зачем. Во-первых, он играет эвристическую роль, позволяя догадаться о вычислимости функции на машине Поста до построения вычисляющей ее программы. Во-вторых, он придает математической теории вычислимых на машине Поста функций философское значение, превращая ее в общую теорию вычислимых функций. Сам Пост следующим образом характеризует это значение (см. с. 88 настоящей книги): «Сделано фундаментальное открытие, касающееся математизационных способностей Homo Sapiens».

§ 5. Машина Поста и электронные вычислительные машины

В чем сходство и в чем различие между машиной Поста и современной электронной вычислительной машиной (ЭВМ)?

Сперва займемся сходством, которое имеет место по крайней мере в следующих трех отношениях.

1. Как и в машине Поста, в ЭВМ можно выделить «атомарные» (т. е. далее неделимые) носители инфор-

мации (в машине Поста такими атомарными носителями служат секции ленты): как и в машине Поста, каждый такой атомарный носитель может находиться в одном из двух состояний (в машине Поста — «пустом» или «отмеченном») ; вся информация, хранящаяся в машине в данный момент, предстает в виде распределения этих двух состояний по элементарным носителям.

2. Как и для машины Поста, для ЭВМ указывается некоторый ограниченный набор элементарных действий; за один шаг ЭВМ, как и машина Поста, может совершить какое-либо действие из этого набора.

3. Как и машина Поста, ЭВМ работает на основе особой инструкции — программы, указывающей, какие элементарные действия и в каком порядке должны быть совершены.

Следовательно, чтобы записать в машине информацию, надо отождествить ее с некоторой комбинацией состояний элементарных носителей. А чтобы реализовать алгоритм на машине, надо представить его в виде последовательности определенных элементарных действий. В этом и состоит искусство составителя программ — программиста. А искусство проектировщика машины состоит, в частности, в снабжении машины таким набором элементарных действий, чтобы алгоритмы, предназначенные для реализации на этой машине, удобно расчленялись на эти действия.

Вместе с тем мы хотим обратить внимание читателя на следующие существенные черты ЭВМ, которые либо совершенно отсутствуют, либо не проявляются должным образом в машине Поста.

1. Информация, хранимая в машине Поста, расположена в «запоминающем устройстве» (т. е. устройстве, предназначенном для хранения информации; в случае машины Поста таким устройством является лента) линейно. Это значит, что каждая секция ленты имеет только двух «соседей», к обработке которых машина может перейти после обработки данной секции. В ЭВМ тот или иной «участок» запоминающего устройства может иметь, вообще говоря, гораздо больше «соседних» участков — соседних в том смысле, что машина может перейти к обработке любого из них после обработки исходного участка. Преимущества такой организации запоминающего устройства очевидны, хотя бы с точки зрения сокращения числа шагов работы машины — ведь в машине Поста каретка, чтобы перейти от обозрения одной сек-

ции к обозрению другой, должна непременно пройти все промежуточные секции. Если говорить более подробно, то надо сказать, что запоминающее устройство ЭВМ состоит обычно из двух устройств — «внешнего» запоминающего устройства большой емкости, хранящего информацию линейно, подобно ленте машины Поста, и «внутреннего» запоминающего устройства сравнительно небольшой емкости, но с более богатыми «связями соседства» между отдельными участками этого устройства.

2. В случае машины Поста мы ничего не говорили о том, как именно обеспечивается выполнение программы. Можно мыслить себе, например, что рядом с машиной Поста стоит человек, читает написанную на бумаге программу и, следуя ей, передвигает каретку, печатает и стирает метку. Существенно подчеркнуть, что в ЭВМ выполнение программы происходит *автоматически.* Программа для ЭВМ закодирована специальным образом и записывается так, чтобы она была доступна для «восприятия» машины. Часто такой доступной машине формой записи является запись в виде системы дырок, пробиваемых на ленте (перфоленте) или карточках (перфокартах). После того как изготовлена перфолента (или перфокарты) с программой, работа ЭВМ происходит под управлением этой перфоленты (или этих перфокарт) без дальнейшего вмешательства человека. (Разумеется, нетрудно создать устройство, обеспечивающее автоматическую, без участия человека, работу машины Поста.) В первом приближении работу ЭВМ можно сравнить с работой механического фортепьяно, в котором последовательность нажатия клавиш точно определена записанной на перфоленте «программой» и осуществляется без участия человека.

Замечание. Однако аналогия между ЭВМ и механическим фортепьяно имеет место лишь в первом приближении. В самом деле, работа механического фортепьяно, по существу, мало чем отличается от работы шарманки. В шарманке выступы и углубления валика открывают, при его вращении, каналы звучащих трубок; каждое такое открытие (т. е. открытие канала данной трубки в данный момент) управляется своей неровностью на поверхности валика; число звуков в мелодии равно числу этих неровностей, чем длиннее мелодия, тем больше должен быть валик. В механическом фортепьяно каждое нажатие клавиши вызывается некоторым участком перфоленты; следовательно, чем длиннее мелодия,

тем длиннее должна быть программа, записанная на перфоленте. И в шарманке и в механическом фортепьяно запись на валике (посредством выступов и углублений на его поверхности) и на перфоленте (посредством дырок) есть своего рода нотная запись. Иначе обстоит дело в ЭВМ. Здесь, как и в машине Поста, одна и та же команда программы может выполняться *много раз*. Поэтому число шагов работы машины не связано с длиной программы: одна и та же программа может обеспечить любое нужное для достижения результата число шагов. (Например, программа, указанная нами в качестве решения задачи 2 из главы «Прибавление единицы на машине Поста», имеет длину 4, а число шагов, нужное для получения требуемого в этой задаче результата, зависит от того, где вначале стоит каретка.) Это важнейшее свойство и дает возможность реализовать на машине алгоритмы, а именно, составлять *единые программы*, приводящие к результату для целого класса возможных исходных данных. Это свойство обеспечивается возможностью переходить от исполнения одной команды к исполнению любой другой, в то время как в механическом фортепьяно или шарманке после какой-либо «команды» нажатия клавиш или открывания звучащих каналов может выполняться лишь непосредственно следующая — в записи на валике или перфоленте — «команда».

3. Наиболее существенное, принципиальное отличие ЭВМ от машины Поста состоит в следующем. В запоминающее устройство — на ленту машины Поста — перед началом работы записывается исходное данное (например, кортеж чисел, подлежащих сложению). Программа, которая должна применяться к этому исходному данному и согласно которой работает машина, находится как бы в стороне. В ЭВМ перед началом работы машины программа закладывается в запоминающее устройство вместе с исходным данным. (В этом, кстати, еще одно отличие ЭВМ от механического фортепьяно: в механическом фортепьяно записанная на перфоленте программа «читается» специальным устройством, и по мере чтения происходит игра; это устройство выполняет, таким образом, роль пианиста, играющего по разложенным перед ним нотам. В ЭВМ перфолента с записанными на ней программой и исходным данным перед началом работы машины обрабатывается специальным «устройством ввода», которое прочитывает содержащуюся на перфо-

ленте информацию и помещает ее в запоминающее устройство; в этом отношении машина подобна пианисту, который сперва выучивает ноты наощупь, а потом играет уже по памяти, не глядя в ноты.) Содержание запоминающего устройства в машине в некоторый момент — перед началом работы — будем называть начальной информацией. В машине Поста начальная информация состоит только из исходного данного. В ЭВМ начальная информация состоит из исходного данного и программы. Важно подчеркнуть, что разделение начальной информации в ЭВМ на исходное данное и программу довольно условно. Вся эта информация, включая и ту ее часть, которая названа «программой», может подвергаться различным преобразованиям в процессе работы машины. Эта возможность изменять сами команды в процессе работы, отсутствующая в машине Поста, является очень важной чертой современных ЭВМ. С другой стороны, содержание более или менее любого участка запоминающего устройства очень часто можно интерпретировать как команду. Поэтому разделение информации, содержащейся в запоминающем устройстве, на программу и исходное данное вообще имеет смысл только в начальный момент, затем на каждом шаге вся эта информация в целом подвергается переработке.

Естествен вопрос: по какому же закону происходит переработка информации, содержащейся в запоминающем устройстве? Эта переработка происходит в свою очередь на основе некоторого правила, которое мы будем называть Универсальной Программой. Универсальную Программу ни в коем случае нельзя смешивать с частной программой, предназначенной для решения той или иной конкретной задачи и применяющейся к исходным данным. Частная программа зависит от решаемой задачи, в то время как Универсальная Программа одна для всех задач; в существующих ЭВМ она заложена в самой конструкции машины (поэтому такие машины часто называют *универсальными*). Подытожим сказанное. В машине Поста вначале в запоминающее устройство закладывается исходное данное, после чего работа происходит по программе, не находящейся в запоминающем устройстве. В ЭВМ вначале в запоминающее устройство закладывается начальная информация, слагающаяся из частной программы решения данной задачи и исходного данного, после чего работа (а именно, переработка всего содержания запоминающего устройства)

происходит по Универсальной Программе, воплощенной в конструкции машины [1]).

Могут указать еще на одно, казалось бы, очень существенное отличие ЭВМ от машины Поста: запоминающее устройство машины Поста — лента — имеет бесконечную емкость, чего не может быть в реальных машинах. Однако и в машине Поста можно заменить бесконечную ленту конечной лентой, подклеиваемой по мере надобности (ведь все равно к каждому моменту работы машины Поста будет использован только конечный кусок ленты): когда каретка доходит до конца ленты, к нему подклеивается еще несколько секций. С другой стороны, и внешнее запоминающее устройство ЭВМ может в принципе неограниченно увеличивать свою емкость путем добавления новых его частей (скажем, новых магнитных лент). Таким образом, и машину Поста и ЭВМ можно трактовать как обладающие запоминающим устройством, конечным в каждый данный момент, но неограниченно растущей емкости. Именно это обстоятельство, делающее указанные машины пригодными для осуществления на них любого алгоритма, главным образом и роднит машину Поста с ЭВМ и позволяет рассматривать машину Поста как упрощенную модель ЭВМ.

---

[1]) Можно, впрочем, составить Универсальную Программу и для машины Поста, а именно можно закодировать каждую программу машины Поста в виде пóстова слова и помещать далее запись этой программы (т. е. запись соответствующего слова) на ленте рядом с записью того исходного данного, к которому она должна применяться. Можно, далее, составить программу — Универсальную Программу, — которая, будучи применена к записи, составленной из записи некоторой программы П и записи некоторого исходного данного x, давала бы тот же результат, что и непосредственное применение П к записи x. Однако в этом случае Универсальная Программа будет всего лишь одной из возможных программ машины Поста: в случае же ЭВМ Универсальная Программа вовсе не является одной из частных программ, допустимых для ЭВМ, а реализована самой структурой ЭВМ.

ПРИЛОЖЕНИЕ 1

В качестве приложения мы предлагаем перевод знаменитой статьи Эмиля Поста «Финитные комбинаторные процессы, формулировка 1» (Emil L. Post «Finite combinatory processes — formulation 1»). Статья эта была опубликована в 1936 г. в 3-м, сентябрьском номере 1-го тома «Журнала символической логики («The Journal of Symbolic Logic»), выходящего ежеквартально. Публикация сопровождалась следующим примечанием редакции журнала: «Получено 7 октября 1936 г. Читателю рекомендуется сравнить статью А. М. Тьюринга «О вычислимых числах», долженствующую появиться вскоре в «Трудах Лондонского математического общества». Настоящая статья, однако, хотя и имеет более позднюю дату, написана совершенно независимо от статьи Тьюринга». Статья Тьюринга (имеющая дату поступления 28 мая 1936 г.) была опубликована в том же 1936 г. (Turing A. M. On computable numbers, with an application to the Entscheidungsproblem//Proc. London Math. Soc., ser. 2. — 1936. — V. 42. — № 3—4, — P. 230—265. A correction//Proc. London Math. Soc., ser. 2. — 1937. — V. 43. — № 7. — P. 544—546). Итак,

ФИНИТНЫЕ КОМБИНАТОРНЫЕ ПРОЦЕССЫ, ФОРМУЛИРОВКА 1

Эмиль Л. Пост

Предлагаемая формулировка может представить интерес при развитии символической логики в направлении, намеченном теоремой Гёделя о неполноте символических логик [1] и результатом Чёрча относительно абсолютно неразрешимых проблем [2].

---

[1] Gödel Kurt. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I.//Monatsh. Math. Phys. — 1931. — Bd. 38. — H. 1. — S. 173—198. [Все подстрочные примечания к статье Э. Л. Поста принадлежат ее автору. — *Примеч. пер.*]

[2] Church. Alonzo. An unsolvable problem of elementary number theory//Amer. J. Math. — 1936. — V. 58. — № 2. — P. 345—363.

Мы имеем в виду *общую проблему,* состоящую из множества *конкретных проблем.* Решением общей проблемы будет такое решение, которое доставляет ответ для каждой конкретной проблемы.

В излагаемой ниже формулировке того, что есть решение, используются два понятия: *пространство символов,* в котором должна производиться работа, приводящая от проблемы к ответу[3], и зафиксированный неизменяемый *набор инструкций,* который будет и управлять операциями в пространстве символов и определять порядок, в котором эти инструкции должны применяться.

В нашей формулировке пространство символов состоит из бесконечной в обе стороны последовательности мест, или ящиков; с точки зрения порядка, таким образом, оно подобно ряду целых чисел ..., —3, —2, —1, 0, 1, 2, 3, ... Решатель проблемы, или работник, может передвигаться и работать в этом пространстве символов, будучи в состоянии в каждый отдельный момент находиться и действовать лишь в одном ящике. Не считая присутствия работника, каждый ящик может принимать лишь одно из двух возможных состояний, а именно: либо быть пустым, или непомеченным, либо же нести на себе некоторую единую метку,— скажем, вертикальную черту.

Один из ящиков должен быть выделен; он называется отправным пунктом. Мы предполагаем далее, что конкретная проблема должна задаваться в символической форме посредством конечного числа помеченных ящиков. Аналогичным образом, и ответ предъявляется в символической форме, посредством подобной же конфигурации помеченных ящиков. Более конкретно, ответ представляет собой ту конфигурацию помеченных ящиков, которая возникает по завершении процесса решения.

Предполагается, что работник может совершать следующие примитивные операции[4]:

(a) пометить ящик, в котором он находится (если тот пуст),

(b) стереть метку из ящика, в котором он находится (если тот помечен).

(c) переместиться в ящик справа,

(d) переместиться в ящик слева,

(e) определить, помечен или нет тот ящик, в котором он находится.

[3] Как в пространстве символов, так и во времени.

[4] Равно как и следовать инструкциям, описываемым ниже.

Набор инструкций (который, следует подчеркнуть, является одним и тем же для всех конкретных проблем и тем самым соотнесен с общей проблемой в целом) должен иметь следующую форму. Он начинается так:

Отправляйся от отправного пункта и следуй инструкции 1.

Далее набор содержит конечное число инструкций, занумерованных числами 1, 2, 3, ..., n. При этом i-я инструкция должна иметь один из следующих видов:

(A) *соверши операцию* O_i [O_i = (a), (b), (c) или (d)] *и затем следуй инструкции* j_i,

(B) *соверши операцию* (e) *и в зависимости от того, будет ответ «да» или «нет», следуй инструкции* j_i' *или* j_i'',

(C) *остановись.*

Ясно, что достаточно иметь лишь одну инструкцию типа (C). Заметим еще, что состояние пространства символов непосредственно влияет на процесс только посредством инструкций типа (B).

Будем говорить, что набор инструкций *применим* к данной общей проблеме, если его применение к каждой частной проблеме никогда не потребует операции (a), когда ящик, в котором находится работник, помечен, или операции (b), когда ящик не помечен[5]. Набор инструкций, применимый к общей проблеме, задает — при применении к каждой конкретной проблеме — детерминированный процесс. Этот процесс закончится тогда и только тогда, когда он дойдет до инструкции типа (C). Будем говорить, что набор инструкций задает *финитный 1-процесс* в связи с данной общей проблемой, если он применим к этой проблеме и если *определяемый набором процесс заканчивается для каждой конкретной проблемы*: Финитный 1-процесс, связанный с некоторой общей проблемой, будем называть *1-решением* этой проблемы, если даваемый им для каждой конкретной проблемы ответ всегда является правильным.

Мы не занимаемся здесь тем, каким образом конфигурация помеченных ящиков, соответствующая конкретной проблеме или ответу на нее, символически изображает осмысленные проблему и ответ. Сказанное выше предполагает в действительности, что конкретная проблема задается в символизированной форме некоторой внешней силой и на аналогичной основе воспринимается

[5] Хотя наша формулировка набора инструкций и может быть легко перередактирована так, что применимость будет гарантирована, это представляется нежелательным по целому ряду причин.

символический ответ. Усовершенствование, делающее построения более автономными, таково. Общая проблема состоит, очевидно, из не более чем счетно бесконечного множества конкретных проблем. Не будем рассматривать конечный случай. Пусть между классом положительных целых чисел и классом конкретных проблем установлено взаимно однозначное соответствие. Мы можем, довольно-таки произвольно, изобразить целое положительное n, пометив n первых ящиков направо от отправного пункта. Общую проблему будем называть *1-заданной*, если учрежден такой финитный 1-процесс, который, будучи применяем к классу положительных целых чисел, символически изображенных, как только что указано, выдает, причем взаимно однозначным образом, класс конкретных проблем, образующий рассматриваемую общую проблему. Примем для удобства, что когда общая проблема 1-задана указанным способом, каждый конкретный процесс при своем завершении оставляет работника в отправном пункте. Если теперь некоторая общая проблема 1-задана и 1-разрешима, мы можем, с очевидными изменениями, соединить два набора инструкций и получить финитный 1-процесс, который дает ответ на каждую конкретную проблему, коль скоро последняя задана просто своим номером в символическом изображении.

С некоторой модификацией изложенная формулировка приложима также и к символическим логикам. Мы должны теперь иметь дело не с классом конечных проблем, но с одним-единственным начальным конечным распределением меток по пространству символов, изображающим символически исходные формальные утверждения логики. С другой стороны, теперь у нас не будет инструкций типа (C). Следовательно, в предположении применимости, задается детерминированный процесс, являющийся *нескончаемым*. Мы предполагаем, далее, что в течение процесса появляются некоторые могущие быть опознанными группы символов — т. е. конечные последовательности помеченных и непомеченных ящиков,— которые далее в течение процесса не изменяются. Они будут представлять собой выводимые утверждения логики. Набор инструкций, конечно, соответствует дедуктивным процессам логики. Логика может быть названа в таком случае *1-порождаемой*.

Альтернативная процедура, менее, однако, гармонирующая с духом символической логики, могла бы со-

стоять в указании финитного 1-процесса, который выдает n-ю теорему, или n-е формальное утверждение, логики, коль скоро предъявлено n — опять-таки в разъясненном выше символическом изображении.

Наше исходное представление о заданной конкретной проблеме приводит к трудности, о которой стоит упомянуть. А именно, коль скоро внешняя сила задает начальную конечую разметку пространства символов, для нас не существует способа определить, который из ящиков является первым отмеченным, а который — последним. Эта трудность полностью устраняется в случае, если общая проблема 1-задана. Она оказывается успешно обойденной также всякий раз, как указан финитный 1-процесс. На практике осмысленные конкретные проблемы могут быть изображены символически таким образом, что границы соответствующего символического изображения опознаются посредством характерных групп помеченных и непомеченных ящиков.

Однако корень нашего затруднения лежит, вероятно, в допущении бесконечного пространства символов. В настоящей формулировке ящики суть, по крайней мере умозрительно, физические сущности, например, смежные клетки (поля). Наша внешняя сила может не в большей степени предъявить нам бесконечное число таких ящиков, чем пометить бесконечное число из уже предъявленных. Если же эта сила преподносит нам конкретную проблему в виде конечного куска пространства символов, обсуждаемая трудность исчезает. Разумеется, это потребовало бы расширения перечня примитивных операций с тем, чтобы сделать возможным необходимое расширение исходного пространства символов по мере развития процесса. Таким образом, окончательный вариант формулировки этого типа должен был бы содержать также инструкции для порождения пространства символов[6].

Автор ожидает, что его формулировка окажется логически эквивалентной рекурсивности в смысле Гё-

[6] Развитие формулировки 1 склонно становиться на своих начальных стадиях довольно сложным. Как это ни дисгармонирует с духом подобных формулировок, окончательная ее форма может пожертвовать ее теперешней простотой в пользу большей гибкости. Одна из возможностей состоит в том, чтобы иметь более одного способа помечать ящик. Не исключено, что желаемая естественность может быть достигнута путем допущения конечного числа, возможно двух, физических объектов, предназначенных для использования в качестве указателей, с тем чтобы работник мог опознавать их и передвигать из ящика в ящик.

деля — Чёрча[7]. Цель формулировки, однако, в том, чтобы предложить систему не только определенной логической силы, но и психологической достоверности. В этом последнем смысле подлежат рассмотрению все более и более широкие формулировки. С другой стороны, нашей целью будет показать, что все они логически сводимы к формулировке 1. В настоящий момент мы выдвигаем это умозаключение в качестве *рабочей гипотезы.* Таково же, по нашему мнению, и предложенное Чёрчем отождествление эффективной вычислимости с рекурсивностью[8]. Из этой гипотезы — и вследствие ее видимого контраста со всем математическим развитием, начиная с канторова доказательства несчетности точек на прямой, — независимо вытекают результаты в стиле Гёделя — Чёрча. Успех вышеизложенной программы заключался бы для нас в превращении этой гипотезы не столько в определение или аксиому, сколько в закон природы. Только так, представляется автору, теорема Гёделя относительно неполноты символических логик некоторого общего типа и результат Чёрча о рекурсивной неразрешимости некоторых проблем могут быть преобразованы в умозаключения, относящиеся ко всем символическим логикам и всем методам решения.

Колледж города Нью-Йорка

---

[7] Возможно, что требуемое сопоставление было бы легче всего осуществить, определив понятие 1-функции и доказав эквивалентность этого определения определению рекурсивной функции. (См. указанную в примечании[2] работу Чёрча, с. 350.) Под 1-функцией понималась бы при этом такая функция $f(n)$ с целыми положительными аргументами и значениями, для которой может быть указан финитный 1-процесс со следующим свойством: для каждого целого положительного n, взятого в качестве проблемы, процесс дает $f(n)$ в качестве ответа, притом что n и $f(n)$ изображаются символически, как договорено выше.

[8] Ср. указанную в примечании[2] работу Чёрча, с. 346, 356—358. Впрочем, работа, проведенная Чёрчем и другими, относит это отождествление далеко за пределы рабочей гипотезы. Но попытка маскировать это отождествление определением скрывает тот факт, что сделано фундаментальное открытие, касающееся математизационных возможностей Homo Sapiens, и делает нас слепыми в отношении необходимости постоянно подтверждать это отождествление.

ПРИЛОЖЕНИЕ 2

РЕШИМЫЕ И НЕРЕШИМЫЕ АЛГОРИТМИЧЕСКИЕ ПРОБЛЕМЫ*)

В. А. Успенский, А. Л. Семенов

Алгоритмы. Все мы умеем складывать многозначные числа — этому учат в детстве. Давайте задумаемся, в чем состоит наше умение? Когда в пьесе Э. Ионеско «Урок» ученица демонстрирует замечательные способности к перемножению десятизначных чисел, потрясенный учитель узнает, что ее способ очень прост — она выучила (бесконечную!) таблицу умножения всех чисел. По-видимому, наш способ — иного рода. Мы знаем единый конечный способ, позволяющий решать бесконечное множество задач. В нашем случае бесконечное множество образуют задачи о нахождении суммы: для каждой пары чисел — своя задача. Способ единый, поскольку он один и тот же для всех задач. Способ конечный в том смысле, что он может быть изложен конечным числом слов (а если бы это было не так, то ведь этому способу нельзя было бы научить!). Как известно, способ этот называется «сложением столбиком» и состоит в том, что одно число (например, 8357) подписывается снизу под другим (например, 64) с выравниванием правого края (то есть так: $\begin{matrix}64\\8357\end{matrix}$, а не, скажем, так: $\begin{matrix}64\\8357\end{matrix}$) и далее начинается последовательное применение таблицы сложения однозначных чисел (которую нужно заранее выучить) с переносом, где надо, в следующий разряд. В результате этого постепенно будет сформировано число 8421.

Здесь мы не столько изложили способ, сколько напомнили его, будучи уверены, что он известен. Однако читателю было бы полезно записать этот способ на бумаге полностью — так, чтобы его мог понять и использовать человек, не знавший ранее, как складывать многозначные числа. Упражнение, которое мы предложили читателю, на научном языке формулируется так: записать алгоритм, решающий проблему сложения чисел. Эту и другие проблемы построения алгоритмов мы будем далее формулировать, указывая, что от алгоритма требуется, примерно в таком виде:

Проблема. *Дано: два произвольных натуральных числа. Требуется: найти их сумму.*

Такая форма записи будет служить для нас сигналом, что решение проблемы состоит не в том, чтобы для каждых двух чисел найти свой отдельный способ вычисления результата, а в том, чтобы найти единый общий способ, годящийся для всех пар чисел, то есть найти алгоритм (в данном случае — алгоритм сложения).

Алгоритмом называется всякая конечная система правил, которая задает вычислительный процесс. Этот процесс начинается с предъявления (*подачи на вход* алгоритма) произвольного (но выбранного из фиксированной для данного алгоритма совокупности) *исходного данного.* Процесс может завершиться получением полностью определенного этим исходным данным результата; тогда го-

*) Перепечатка с небольшим сокращением одноименной статьи В. А. Успенского, А. Л. Семенова из журнала «Квант» (1985, № 7, с. 9—15).

ворят, что результат образуется *на выходе* алгоритма. Однако процесс может продолжаться и неограниченно, у алгоритма результат существует не всегда. Например, алгоритм, выписывающий частное двух чисел в виде десятичной дроби, может никогда не кончить работу (так $1 : 3 = 0{,}33333\ldots$).

Понятие алгоритма принадлежит к числу основных понятий математики. Теория алгоритмов вместе с математической логикой образует фундаментальную теоретическую базу построения и использования вычислительной техники. Именно наличие алгоритмов для тех или иных задач позволяет решать эти задачи на ЭВМ.

Замечателен сам термин «алгоритм». В самом деле, много ли терминов науки, в состав которых входит географическое название? Подобное название не сразу разглядишь в слове «алгоритм», однако оно там есть. Это название древнего государства Средней Азии — Хорезм. Ведь в основе термина «алгоритм» лежит имя ал-Хорезми (что значит «из Хорезма») — великого математика IX века. Крупнейшие математики и алгоритмисты всего мира воздали дань основополагающей роли ал-Хорезми, съехавшись в сентябре 1979 года в Хорезмскую область Узбекистана, в город Ургенч, где они обсудили насущные вопросы теории алгоритмов и ее приложений на современном этапе. Подробнее об этой встрече в Ургенче рассказал один из ее организаторов, академик А. П. Ершов (см. «Квант», 1984, № 8, с. 29).

Мы собираемся продемонстрировать читателю наиболее знаменитые и наглядные примеры *нерешимых* алгоритмических проблем, то есть таких алгоритмических проблем, для которых не существует решения. Для контраста, эти примеры будут соседствовать с такими, где алгоритм есть (хотя, иногда, и очень сложен) или вопрос о наличии алгоритма еще не решен.

Решение алгебраических уравнений. Выпишем уравнение

$$x^2 + 2 = y^3 \qquad (1)$$

и будем интересоваться его решениями. Решение в данном случае — это пара чисел $(a; b)$, подчиненная условию $a^2 + 2 = b^3$, например, пара $(5; 3)$.

Решая какое-либо уравнение, надо всегда точно знать, какие числа дозволяется брать в качестве его решений. Если, к примеру, решением уравнения должен служить прирост количества коз в таком-то стаде за такой-то срок, вряд ли нас устроит ответ «две трети» (но ответ «минус десять» будет осмыслен: он означает, что количество коз в стаде, к сожалению, не возросло, а уменьшилось). Для уравнения (1) можно интересоваться его действительными решениями, то есть парами действительных чисел, рациональными решениями, то есть парами рациональных чисел, целыми решениями, то есть парами целых чисел. Уравнение (1) имеет бесконечно много действительных решений: достаточно взять любое a и положить $b = \sqrt[3]{a^2 + 2}$, тогда пара $(a; b)$ будет решением (вопрос читателю: можно ли взять любое b и положить $a = \sqrt{b^3 - 2}$?). Нетрудно доказать, что (1) имеет бесконечно много рациональных решений; вот одно из них: $(0{,}383; 0{,}129)$. Целых же решений только два: $(5; 3)$ и $(-5; 3)$. Однако доказать это нелегко.

Решение уравнений в целых числах представляет собой очень древнюю (ею занимался еще сам Пифагор в VI в. до н. э.) и — в общем случае — очень трудную задачу. Основная трудность состоит в том, чтобы выяснить, имеет ли данное уравнение хотя бы

одно целочисленное решение. Например, неизвестно (во всяком случае было неизвестно до сравнительно недавнего времени) имеют ли решения в целых числах уравнения $x^3+y^3+z^3=30$ и $x^4+y^4+z^4=u^4$.

Задача для читателей. Для каждого из уравнений $x^4+y^4=5z^2$, $3x^2-y^2=1$, $x^2+y^3=z^4$ найдите какое-нибудь решение в целых числах или докажите, что такого решения не существует.

Указание. Об этих и других уравнениях в целых числах можно прочесть в популярных брошюрах: Гельфонд А. О. «Решение уравнений в целых числах», М., 1984 (серия «Популярные лекции по математике», вып. 8); Серпинский В. «О решении уравнений в целых числах», М., 1961.

Задача «уметь по произвольному алгебраическому уравнению с целыми коэффициентами выяснять, имеет ли оно решение в целых числах» составляет одну из двадцати трех так называемых *проблем Гильберта.* Что это за проблемы?

Когда Давид Гильберт (1862—1943), один из величайших математиков XX века, готовил свой доклад на Всемирном математическом конгрессе в Париже в 1900 г., он выбрал 23 проблемы из различных областей математики. Эти проблемы были его попыткой «проникнуть в предстоящие успехи нашей науки и тайны ее развития в начинающемся столетии». Попытка эта оказалась необычайно успешной и ценной. И по сей день решение (даже частичное) какой-либо проблемы Гильберта расценивается во всем мире как высшее математическое достижение. Обязательным требованием, предъявляемым к действительно фундаментальной проблеме, Гильберт считал «возможность изложить ее содержание первому встречному». Задача о выяснении наличия целочисленных решений составляет десятую проблему из списка Гильберта. Читатель может сам судить, в какой степени ее формулировка является общедоступной. Сейчас мы сформулируем десятую проблему Гильберта в соответствии с принятыми нами стандартами. Напомним, что *алгебраическим* называется уравнение вида $P(x_1, \ldots, x_n)=0$, где P — многочлен.

Проблема 1 (*десятая проблема Гильберта*). *Дано: произвольное алгебраическое уравнение с целыми коэффициентами. Требуется: выяснить, существует ли у данного уравнения решение в целых числах.*

Искомый алгоритм: в 1970 г. советский математик Ю. В. Матиясевич доказал невозможность алгоритма, решающего эту проблему (см. «Квант», 1970, № 7, с. 39).

В связи со сформулированной проблемой уместно сделать замечание, применимое не только к ней, но, с соответствующими изменениями, и в других случаях. Алгоритм отыскания какого-нибудь решения любого уравнения, о котором известно, что у него есть решение в целых числах, существует! Действительно, вот такой алгоритм: надо просто подставлять в уравнение все наборы целых чисел. В конце концов будет найден набор, удовлетворяющий уравнению.

Приведем несколько естественных модификаций первой проблемы.

Проблема 2. *Дано: произвольное алгебраическое уравнение с целыми коэффициентами. Требуется: выяснить, существует ли у данного уравнения решение в действительных числах.*

Искомый алгоритм: существует!

Решение проблемы 2 было получено известным польским логиком и математиком Альфредом Тарским в 40-е годы XX века. О применении этого результата будет говориться дальше. В дей-

ствительности, Тарский построил более общий алгоритм, позволяющий выяснить наличие решений у систем равенств и неравенств.

Чтобы оценить степень нетривиальности алгоритма Тарского, попытайтесь решить следующую задачу. По произвольному алгебраическому уравнению с целыми коэффициентами и одной неизвестной требуется выяснить, существует ли у него решение в действительных числах. (Ясно, что эта задача — простой частный случай проблемы 2.) Если степень уравнения нечетна, сразу можно ответить, что корень уравнения существует. А что, если четна?

Проблема 3. *Дано: произвольное алгебраическое уравнение с целыми коэффициентами. Требуется: выяснить, существует ли у данного уравнения решение в рациональных числах.*

Искомый алгоритм: существует он или нет — неизвестно.

Просмотрите, как близки друг к другу формулировки наших проблем 1, 2 и 3: эти формулировки различаются лишь в одном слове. И как, тем не менее, различны ответы на вопрос, существует или нет соответствующий алгоритм. Поистине, удивительное рядом!

Преобразование слов. В математике *словами* называются любые конечные последовательности символов. Например, 1985 — слово длины 4. Слова не только не обязаны обладать каким-либо смыслом, но даже быть произносимыми. Так, БРДРОЬЪЩ — слово из русских букв (и притом длины 8), хотя и не слово русского языка. Как правило, исходные данные для алгоритмов — это слова. Когда алгоритм задан, определено и множество *букв*, то есть символов, из которых составляются слова, подаваемые ему на вход. Такое множество называется *алфави́том*. Алфавиты, в которых пишутся тексты на естественных-языках и языках программирования, печатаются книги, содержат не более нескольких сот букв. Многозначные числа, участвующие в «школьных» алгоритмах — это слова в десятибуквенном алфавите, состоящем из букв 0, 1, 2, ..., 9; эти буквы, как известно, называются цифрами; а для записи десятичных дробей, помимо этих десяти цифр, используется еще одна буква — запятая.

Современные вычислительные машины работают с двоичными словами, состоящими из букв 0 и 1. Чтобы проиллюстрировать гибкость и емкость понятия «слово», заметим, что пару слов можно закодировать одним словом, например, поставив между членами пары букву, не входящую в алфавит, в котором эти члены записаны. Таким образом, в качестве исходных данных для алгоритмов достаточно рассматривать слова, а рассматривать в этом качестве пары, тройки и т. д. слов нет нужды. Заметим еще, что часто бывает удобно рассматривать слово длины ноль; оно называется *пустым словом*, так как вовсе не содержит букв.

Многие из вас играли в следующую игру. Дается задание, например, «переделать волка в козу». Решение:

ВОЛК → ПОЛК → ПОЛА → ПОЗА → КОЗА

В чем состоит один шаг переделки? — В замене любой буквы уже полученного слова на любую другую букву русского алфавита, причем так, чтобы получаемый объект (слово в математическом смысле этого слова) оказывался снова осмысленным словом русского языка.

Теперь упростим игру — забудем о том, что среди всех последовательностей букв выделены слова русского языка. Будем считать все (в математическом понимании, то есть как осмысленные, так и бессмысленные) слова в русском алфавите равноправными. Можно

ли тогда переделать волка в козу? Ну, теперь это совсем легко:

ВОЛК → КОЛК → КОЗК → КОЗА

Может быть, теперь любое слово можно переделать в любое? Конечно нет — слова должны быть одинаковой длины. Но этого условия и достаточно!

Проблема. *Дано: два слова в русском алфавите. Требуется: узнать, получается ли одно из другого последовательностью замен буквы на букву.*

Искомый алгоритм: сравнить длины слов, если они совпадают, то ответ ДА, если не совпадают, то ответ — НЕТ.

Заметим, что первоначальная проблема — переделать одно заданное слово русского языка — также очевидно решима — для ее решения нужно только иметь (конечный!) словарь всех слов русского языка, которые можно использовать в игре, и достаточный запас времени и бумаги. Несложный подсчет показывает, что эта задача может быть практически решена на сегодняшних ЭВМ.

Теперь, опять забыв о русском языке (но не о русском алфавите) и под словами понимая произвольные конечные последовательности русских букв, усложним задачу в другом отношении. Вместо того, чтобы разрешать заменять (в любом месте слова) любую букву на любую, выберем некоторый конечный список разрешенных замен. Этот список может выглядеть, например, так:

А	ББ	БА
↕	↕	↕
Б	АА	АБА

Преобразование слова за один шаг будет, как и раньше, состоять в том, что в любом месте слова мы заменяем участок, который разрешено заменять, на то, на что его разрешено заменять. При этом все разрешенные замены двусторонние: если участок *X* разрешено заменять на участок *Y*, то и участок *Y* разрешено заменять на участок *X*. Например, если список разрешенных замен таков, какой мы только что выбрали, то преобразование за один шаг может быть таким: **БАА**Б — Б**ББББ**Б, или таким: **БА**Б — **АБА**Б, или таким: БА**БА** — БА**А**.

Проблема. *Дано: два слова в русском алфавите. Требуется: узнать, можно ли переделать одно в другое, пользуясь следующей системой разрешенных замен:*

АВ	БВ	АУ	БУ	ДВА	ДУБ	ВВА
↕	↕	↕	↕	↕	↕	↕
ВА	ВБ	УА	УБ	ВД	УД	ВВАД

Искомый алгоритм: не существует. В 1957 году это доказал советский математик Г. С. Цейтин, придумавший указанную систему замен.

Подобную проблему в общем виде (то есть для любой системы разрешенных замен) впервые поставил известный норвежский математик Аксель Туэ в 1914 году. Сам он доказал существование алгоритма в ряде частных случаев.

Туэ особо выделил случаи, когда во всякой разрешенной замене один из членов — пустое слово, и, таким образом, всякая разрешенная замена состоит в том, что можно в любое место имеющегося слова вставить указанное в замене слово или вычеркнуть его.

Например, вставляя и вычеркивая слова АБ и БА, можно из слова АББББА получить БАБАБ. Попытайтесь построить требуемый алгоритм для вставок и вычеркиваний АБ и БА, то есть для такого списка разрешенных замен:

АБ	БА
↓	↓
пустое слово	*пустое слово*

(Уже этот простейший пример важен для раздела алгебры, называемого «теория групп».)

Для любого набора разрешенных вставок-вычеркиваний алгоритм, устанавливающий, можно ли переделать одно слово в другое, существует. Показал это советский математик С. И. Адян в 1966 году.

Элементарная геометрия. Мы не имеем здесь возможности дать детальное определение, что такое элементарная геометрия в строгом смысле, уточняемом с помощью математической логики. Постараемся, однако, дать приблизительное описание. Исходными понятиями элементарной геометрии служат точки, прямые, плоскости, отрезки, окружности, сферы, отношения «лежать на», «лежать между», «быть конгруэнтными» и т. п. Допускается образование новых понятий — но без использования многоточий и оборота «и так далее» (в этом пункте строгое математико-логическое представление об элементарной геометрии расходится с расплывчатым традиционным). Например, понятие треугольника принадлежит элементарной геометрии: ведь треугольник можно определить как такие три отрезка A_1A_2, A_3A_4, A_5A_6, что $A_2 = A_3$, $A_4 = A_5$, $A_6 = A_1$. А вот общее понятие ломаной не принадлежит элементарной геометрии, потому что в его определении участвуют многоточие и слова «и так далее»: ломаная — это набор отрезков A_1A_2, A_3A_4, ... (здесь многоточие!), A_{n-1}, A_n, таких, что $A_2 = A_3$, $A_4 = A_5$ и т. д.

Проблема. *Дано: утверждение элементарной геометрии. Требуется: проверить, истинно ли оно.*

Искомый алгоритм: существует.

Существование алгоритма было, по существу, установлено Тарским в 40-е годы нашего века. Оно достаточно просто вытекает из существования того алгоритма Тарского, о котором говорилось ранее. Связь проблем алгебры и геометрии была открыта уже давно (аналитическая геометрия), естественно, что она распространилась и на алгоритмические проблемы.

Результат Тарского выглядит, и действительно является, крайне сильным и даже парадоксальным. Оказывается, существует весьма обширная область классической математики (именно — элементарная геометрия), о которой традиционно считается, что она требует тонкого интеллекта и уж никак не сводится к тупым подсчетам, и в то же время имеется алгоритм, который может проверять истинность теорем этой области (а если нужно, то и искать их доказательства). Утешиться можно лишь тем, что алгоритм Тарского не является реальным: он требует неслыханно большого времени для своего осуществления.

Полезно представить себе: насколько «близко» начинаются те геометрические утверждения, для которых, несмотря на существование общего алгоритма, сейчас неизвестна их истинность или ложность.

Вот один наглядный пример. Возьмем двенадцать круглых банок одинакового диаметра (равного, скажем, 10 см). Мы хотим упаковать их, поставив стоймя, в круглую коробку. Это можно сделать, например, так, как показано на рисунке 1). Однако можно расположить банки и более тесно — так, чтобы понадобилась коробка меньших размеров. Например так, как показано на рисунке 2. Наконец, упаковка, показанная на рисунке 3, позволяет использовать еще меньшую коробку. Спрашивается, можно ли найти круглую коробку еще меньшего диаметра, в которую бы влезли все наши банки? Ответа на этот вопрос никто не знает, хотя этот ответ

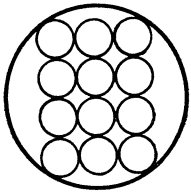


Рис. 1

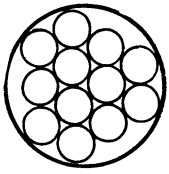


Рис. 2

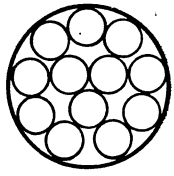


Рис. 3

и может быть получен с помощью алгоритма Тарского: надо применить этот алгоритм к утверждению, что третья упаковка — наилучшая, и посмотреть, что получится в результате. ДА или НЕТ. Однако этот способ нереален, поскольку для получения результата работы алгоритма требуется астрономически большое время.

Распознавание свойств программ. Предположим, что мы работаем в Вычислительном Центре. В этом центре имеется замечательная Вычислительная Машина. Она может производить вычисления по любой программе p, написанной на неком Языке Программирования, то есть по входному данному m находить результат $p(m)$ программы p (если, конечно, он определен).

Программы можно перенумеровать (например, в алфавитном порядке, ведь каждую программу можно считать словом в символах Языка Программирования), составив Библиотеку всех программ:

$$p_1,\ p_2,\ p_3,\ \ldots,\ p_n,\ \ldots$$

Лучше всего иметь дело с такими программами p, которые при любом входном данном m дают результат $p(m)$. Такова, например, стандартная программа для вычисления функции $y = m^2 + 5m - 6$. К сожалению, встречаются и такие программы, которые при некоторых значениях входного данного никогда не заканчивают работы, «зацикливаются», как говорят программисты. Так, программа для вычисления десятичной записи значения функции $y = m/3$ не дает результата, скажем, при $m = 10$.

Это очень неприятное обстоятельство. Ведь если мы такую программу введем в Вычислительную Машину вместе с таким «плохим» исходным данным, работа Вычислительной Машины никогда не окончится и мы будем только зря тратить дорогостоящее Машинное Время.

Чтобы этого избежать, хорошо бы иметь некоторую процедуру (алгоритм!), которая по номеру программы заранее говорила бы, даст программа с этим номером при выбранном входном данном результат, или нет.

Проблема. *Дано: произвольная пара натуральных чисел* (n, m). *Требуется: построить алгоритм, выясняющий, заканчивает ли работу n-я программа* p_n *при входном данном m.*

Искомый алгоритм: не существует.

Давайте подведем некоторые итоги. Мы познакомились с понятием алгоритма — важнейшим понятием современной математики и информатики. Разумеется, многие конкретные алгоритмы (например, алгоритм сложения столбиком или алгоритм Евклида нахождения общей меры) были известны очень давно, однако общее представление об алгоритме сформировалось только в первой половине XX века. Всякая попытка использовать вычислительные машины для решения каких-либо задач начинается с построения алгоритма, решающего эти задачи. Пока нет такого алгоритма, нельзя составлять программу для вычислительной машины. А если оказывается, что нужного алгоритма и не может быть, то бессмысленно и пытаться составлять программу. Поэтому результаты о невозможности тех или иных алгоритмов имеют очевидный практический смысл. Кроме того, рассуждения, приводящие к установлению такой невозможности, позволяют глубже проникнуть в природу самого понятия «алгоритм».

У читателя может возникнуть вопрос: всякий ли алгоритм можно запрограммировать, то есть превратить в программу для вычислительной машины? Ответ: теоретически — да. Слово «теоретически» означает здесь, что ответ утвердителен, если допускать программы сколь угодно длинные (для написания которых заведомо не хватит бумаги, чернил и карандашей на всей Земле) и сколь угодно долго работающие (триллионы лет). Отметим, что подобного рода допущения глубоко коренятся в математике. (Мы допускаем многоугольники с любым числом сторон. Мы верим, что алгоритм сложения столбиком позволяет складывать числа любых размеров, хотя и осознаем, что если в слагаемых больше знаков, чем атомов в Земном шаре, то не вполне ясно, как такие слагаемые записать, да и число шагов при их сложении окажется астрономически велико...) К тому же и возможности вычислительной техники непрерывно растут. Если первые вычислительные машины совершали сотни операций в секунду, и управлялись программами, составленными из десятков команд, то быстродействие современных машин доходит до сотни миллионов операций в секунду, а размер их программ — до сотен тысяч команд. Принято считать, что возможности машин удесятеряются за каждое пятилетие. Разумеется, тем не менее, оценки и размера записи алгоритма и числа шагов его работы имеют первостепенную важность. Но это — тема отдельных серьезных обсуждений.

20 коп.